Introduction to Artificial Intelligence

Introduction Single-Layer Perceptron

Andres Mendez-Vazquez

March 11, 2019

Outline

Cinvestav

Outline

Cinvestav

History

At the beginning of Neural Networks (1943 - 1958)

- 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).

History

At the beginning of Neural Networks (1943 - 1958)

- 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).

In this chapter, we are interested in the perceptron

The perceptron is the simplest form of a neural network used for the classifica tion of patterns said to be linearly separable (i.e., patterns that lie on opposite sides of a hyperplane).

Cinvestav

History

At the beginning of Neural Networks (1943 - 1958)

- 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).

In this chapter, we are interested in the perceptron

The perceptron is the simplest form of a neural network used for the classifica tion of patterns said to be linearly separable (i.e., patterns that lie on opposite sides of a hyperplane).

History

At the beginning of Neural Networks (1943 - 1958)

- 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).

In this chapter, we are interested in the perceptron

The perceptron is the simplest form of a neural network used for the classifica tion of patterns said to be linearly separable (i.e., patterns that lie on opposite sides of a hyperplane).

In addition

Something Notable

- The single neuron also forms the basis of an adaptive filter.
- A functional block that is basic to the ever-expanding subject of signal processing.

In addition

Something Notable

- The single neuron also forms the basis of an adaptive filter.
- A functional block that is basic to the ever-expanding subject of signal processing.

Furthermore

The development of adaptive filtering owes much to the classic paper of Widrow and Hoff (1960) for pioneering the so-called least-mean-square (LMS) algorithm, also known as the delta rule.

In addition

Something Notable

- The single neuron also forms the basis of an adaptive filter.
- A functional block that is basic to the ever-expanding subject of signal processing.

Furthermore

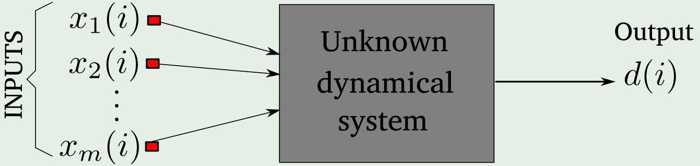
The development of adaptive filtering owes much to the classic paper of Widrow and Hoff (1960) for pioneering the so-called least-mean-square (LMS) algorithm, also known as the delta rule.

Outline

Cinvestav

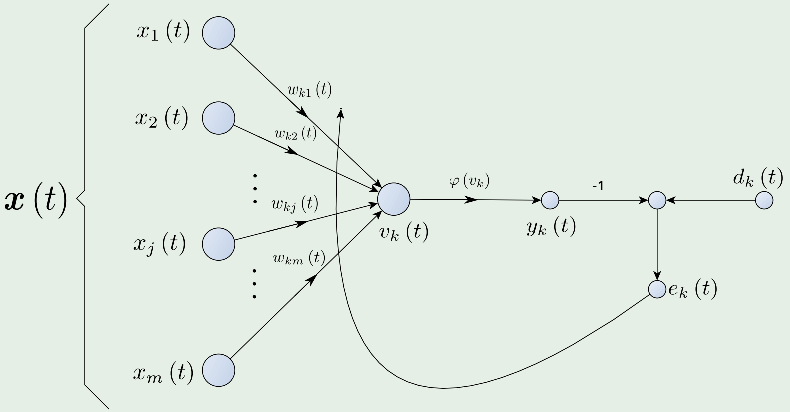
Adapting Filtering Problem

Consider a dynamical system



Signal-Flow Graph of Adaptive Model

We have the following equivalence



Outline

Cinvestav

Description of the Behavior of the System

We have the data set

$$\mathcal{T} = \left\{ \left(\boldsymbol{x}\left(i\right), d\left(i\right) \right) | i = 1, 2, ..., n, ... \right\} \tag{1}$$

Where

$$\boldsymbol{x}\left(i\right) = \left(x_1\left(i\right), x_2\left(i\right) ..., x_m\left(i\right)\right)^T \tag{2}$$

Description of the Behavior of the System

We have the data set

$$\mathcal{T} = \{(\boldsymbol{x}(i), d(i)) \,|\, i = 1, 2, ..., n, ...\} \quad (1)$$

Where

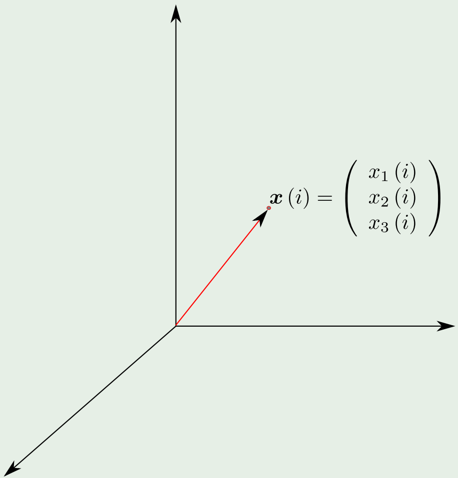
$$\boldsymbol{x}(i) = (x_1(i), x_2(i) ..., x_m(i))^T \quad (2)$$

Cinvestav

The Stimulus $\boldsymbol{x}(i)$

The stimulus $\boldsymbol{x}(i)$ can arise from

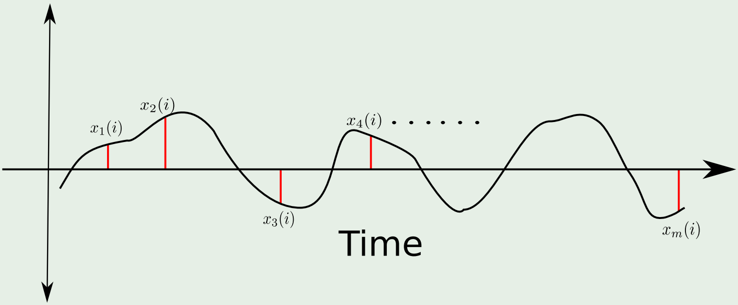
The m elements of $\boldsymbol{x}(i)$ originate at different points in space (spatial)



The Stimulus $\boldsymbol{x}(i)$

The stimulus $\boldsymbol{x}(i)$ can arise from

The m elements of $\boldsymbol{x}(i)$ represent the set of present and $(m-1)$ past values of some excitation that are uniformly spaced in time (temporal).



Problem

Quite important

How do we design a multiple input-single output model of the unknown dynamical system?

It is more

We want to build this around a single neuron!!!

Problem

Quite important

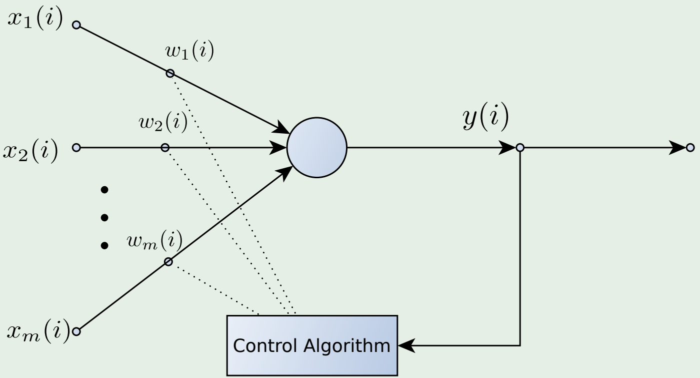
How do we design a multiple input-single output model of the unknown dynamical system?

It is more

We want to build this around a single neuron!!!

Thus, we have the following...

We need an algorithm to control the weight adjustment of the neuron



Which steps do you need for the algorithm?

First

The algorithms starts from an arbitrary setting of the neuron's synaptic weight.

Second

Adjustments, with respect to changes on the environment, are made on a continuous basis.

Which steps do you need for the algorithm?

First

The algorithms starts from an arbitrary setting of the neuron's synaptic weight.

Second

Adjustments, with respect to changes on the environment, are made on a continuous basis.

- Time is incorporated to the algorithm.

Which steps do you need for the algorithm?

First

The algorithms starts from an arbitrary setting of the neuron's synaptic weight.

Second

Adjustments, with respect to changes on the environment, are made on a continuous basis.

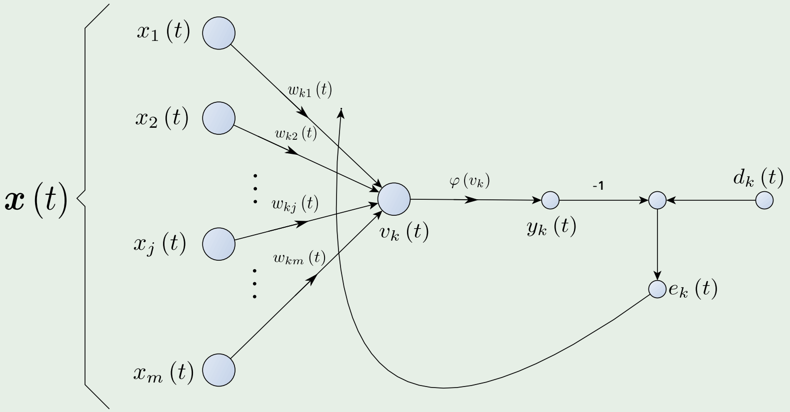
- Time is incorporated to the algorithm.

Third

Computation of adjustments to synaptic weights are completed inside a time interval that is one sampling period long.

Signal-Flow Graph of Adaptive Model

We have the following equivalence



Thus, This Neural Model $\approx$ Adaptive Filter with two continous processes

Filtering processes

1. An output, denoted by $y(i)$, that is produced in response to the m elements of the stimulus vector $\boldsymbol{x}(i)$.
2. An error signal, $e(i)$, that is obtained by comparing the output $y(i)$ to the corresponding desired output $d(i)$ produced by the unknown system.

Adaptive Process

It involves the automatic adjustment of the synaptic weights of the neuron in accordance with the error signal $e(i)$

Remark

The combination of these two processes working together constitutes a **feedback loop** acting around the neuron.

Thus, This Neural Model $\approx$ Adaptive Filter with two continous processes

Filtering processes

1. An output, denoted by $y(i)$, that is produced in response to the m elements of the stimulus vector $\boldsymbol{x}(i)$.
2. An error signal, $e(i)$, that is obtained by comparing the output $y(i)$ to the corresponding desired output $d(i)$ produced by the unknown system.

Adaptive Process

It involves the automatic adjustment of the synaptic weights of the neuron in accordance with the error signal $e(i)$

Remark

The combination of these two processes working together constitutes a **feedback loop** acting around the neuron.

Thus, This Neural Model ≈ Adaptive Filter with two continous processes

Filtering processes

1. An output, denoted by $y(i)$, that is produced in response to the m elements of the stimulus vector $\boldsymbol{x}(i)$.
2. An error signal, $e(i)$, that is obtained by comparing the output $y(i)$ to the corresponding desired output $d(i)$ produced by the unknown system.

Adaptive Process

It involves the automatic adjustment of the synaptic weights of the neuron in accordance with the error signal $e(i)$

Remark

The combination of these two processes working together constitutes a **feedback loop** acting around the neuron.

Thus

The output $y(i)$ is exactly the same as the induced local field $v(i)$

$$y(i) = v(i) = \sum_{i=1}^{m} w_k(i)\, x_k(i) \tag{3}$$

In matrix form, we have - remember we only have a neuron, so we do not have neuron k

$$y(i) = \boldsymbol{x}^T(i)\, \boldsymbol{w}(i) \tag{4}$$

Error

$$e(i) = d(i) - y(i) \tag{5}$$

Thus

The output $y(i)$ is exactly the same as the induced local field $v(i)$

$$y\left(i\right)=v\left(i\right)=\sum_{i=1}^{m}w_{k}\left(i\right)x_{k}\left(i\right) \tag{3}$$

In matrix form, we have - remember we only have a neuron, so we do not have neuron k

$$y\left(i\right)=\boldsymbol{x}^{T}\left(i\right)\boldsymbol{w}\left(i\right) \tag{4}$$

Error

$$e\left(i\right)=d\left(i\right)-y\left(i\right) \tag{5}$$

Thus

The output $y(i)$ is exactly the same as the induced local field $v(i)$

$$y\left(i\right) = v\left(i\right) = \sum_{i=1}^{m} w_k\left(i\right) x_k\left(i\right) \tag{3}$$

In matrix form, we have - remember we only have a neuron, so we do not have neuron k

$$y\left(i\right) = \boldsymbol{x}^T\left(i\right) \boldsymbol{w}\left(i\right) \tag{4}$$

Error

$$e\left(i\right) = d\left(i\right) - y\left(i\right) \tag{5}$$

Cinvestav

Outline

Cinvestav

Consider

A continous differentiable function $J(\boldsymbol{w})$

We want to find an optimal solution $\boldsymbol{w}^*$ such that

$$J(\boldsymbol{w}^*) \leq J(\boldsymbol{w}), \ \forall \boldsymbol{w} \tag{6}$$

We want to

Minimize the cost function $J(w)$ with respect to the weight vector w.

For this

$$\nabla J(w) = 0 \tag{7}$$

Consider

A continous differentiable function $J(\boldsymbol{w})$

We want to find an optimal solution $\boldsymbol{w}^*$ such that

$$J(\boldsymbol{w}^*) \leq J(\boldsymbol{w}), \ \forall \boldsymbol{w} \tag{6}$$

We want to

Minimize the cost function $J(\boldsymbol{w})$ with respect to the weight vector $\boldsymbol{w}$.

For this

$$\nabla J(\boldsymbol{w}) = 0 \tag{7}$$

Cinvestav

Consider

A continous differentiable function $J(\boldsymbol{w})$

We want to find an optimal solution $\boldsymbol{w}^*$ such that

$$J(\boldsymbol{w}^*) \leq J(\boldsymbol{w}),\ \forall \boldsymbol{w} \tag{6}$$

We want to

Minimize the cost function $J(\boldsymbol{w})$ with respect to the weight vector $\boldsymbol{w}$.

For this

$$\nabla J(\boldsymbol{w}) = 0 \tag{7}$$

Cinvestav

Where

∇ is the gradient operator

$$\nabla = \left[\frac{\partial}{\partial w_1}, \frac{\partial}{\partial w_2}, ..., \frac{\partial}{\partial w_m} \right]^T \tag{8}$$

Thus

$$\nabla J(w) = \left[\frac{\partial J(w)}{\partial w_1}, \frac{\partial J(w)}{\partial w_2}, ..., \frac{\partial J(w)}{\partial w_m} \right]^T \tag{9}$$

Where

∇ is the gradient operator

$$\nabla = \left[\frac{\partial}{\partial w_1}, \frac{\partial}{\partial w_2}, ..., \frac{\partial}{\partial w_m} \right]^T \tag{8}$$

Thus

$$\nabla J(\boldsymbol{w}) = \left[\frac{\partial J(\boldsymbol{w})}{\partial w_1}, \frac{\partial J(\boldsymbol{w})}{\partial w_2}, ..., \frac{\partial J(\boldsymbol{w})}{\partial w_m} \right]^T \tag{9}$$

Thus

Starting with an initial guess denoted by $\boldsymbol{w}(0)$,

Then, generate a sequence of weight vectors $\boldsymbol{w}(1), \boldsymbol{w}(2), ...$

Such that you can reduce $J(\boldsymbol{w})$ at each iteration

$$J(\boldsymbol{w}(n+1)) < J(\boldsymbol{w}(n)) \tag{10}$$

Where: $\boldsymbol{w}(n)$ is the old value and $\boldsymbol{w}(n+1)$ is the new value.

Thus

Starting with an initial guess denoted by $\boldsymbol{w}(0)$,

Then, generate a sequence of weight vectors $\boldsymbol{w}(1), \boldsymbol{w}(2), \ldots$

Such that you can reduce $J(\boldsymbol{w})$ at each iteration

$$J(\boldsymbol{w}(n+1)) < J(\boldsymbol{w}(n)) \tag{10}$$

Where: $\boldsymbol{w}(n)$ is the old value and $\boldsymbol{w}(n+1)$ is the new value.

The Three Main Methods for Unconstrained Optimization

We will look at

1. Steepest Descent.
2. Newton's Method
3. Gauss-Newton Method

Outline

Cinvestav

Steepest Descent

In the method of steepest descent, we have a cost function $J(\boldsymbol{w})$ where

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) - \eta \nabla J(\boldsymbol{w}(\boldsymbol{n}))$$

How, we prove that $J(\boldsymbol{w}(n+1)) < J(\boldsymbol{w}(n))$?

We use the first-order Taylor series expansion around $\boldsymbol{w}(n)$

$$J(\boldsymbol{w}(n+1)) \approx J(\boldsymbol{w}(n)) + \nabla J^T(\boldsymbol{w}(\boldsymbol{n}))\Delta \boldsymbol{w}(n) \quad (11)$$

Remark: This is quite true when the step size is quite small!!! In addition, $\Delta \boldsymbol{w}(n) = \boldsymbol{w}(n+1) - \boldsymbol{w}(n)$

Cinvestav

Steepest Descent

In the method of steepest descent, we have a cost function $J(\boldsymbol{w})$ where

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) - \eta \nabla J(\boldsymbol{w}(\boldsymbol{n}))$$

How, we prove that $J(\boldsymbol{w}(n+1)) < J(\boldsymbol{w}(n))$?

We use the first-order Taylor series expansion around $\boldsymbol{w}(n)$

$$J(\boldsymbol{w}(n+1)) \approx J(\boldsymbol{w}(n)) + \nabla J^T(\boldsymbol{w}(\boldsymbol{n}))\Delta\boldsymbol{w}(n) \tag{11}$$

Remark: This is quite true when the step size is quite small!!! In addition, $\Delta\boldsymbol{w}(n) = \boldsymbol{w}(n+1) - \boldsymbol{w}(n)$

Why? Look at the case in $\mathbb{R}$

The equation of the tangent line to the curve $y = J(w(n))$

$$L(w(n)) = J'(w(n))\left[w(n+1) - w(n)\right] + J(w(n)) \tag{12}$$

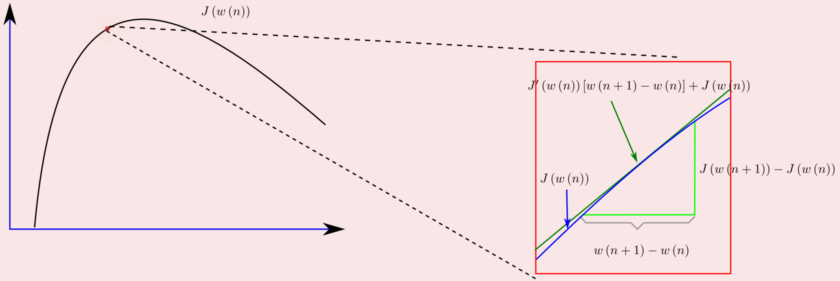
Example

Why? Look at the case in $\mathbb{R}$

The equation of the tangent line to the curve $y = J(w(n))$

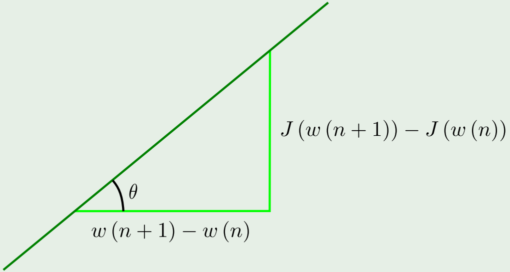
$$L(w(n)) = J'(w(n))[w(n+1) - w(n)] + J(w(n)) \qquad (12)$$

Example



Thus, we have that in $\mathbb{R}$

Remember Something quite Classic



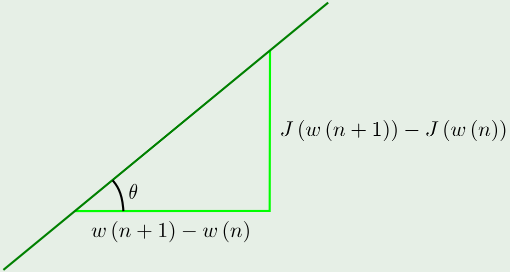
$$\tan\theta = \frac{J(w(n+1)) - J(w(n))}{w(n+1) - w(n)}$$

$$\tan\theta\,(w(n+1) - w(n)) = J(w(n+1)) - J(w(n))$$

$$J'(w(n))\,(w(n+1) - w(n)) = J(w(n+1)) - J(w(n))$$

Thus, we have that in $\mathbb{R}$

Remember Something quite Classic



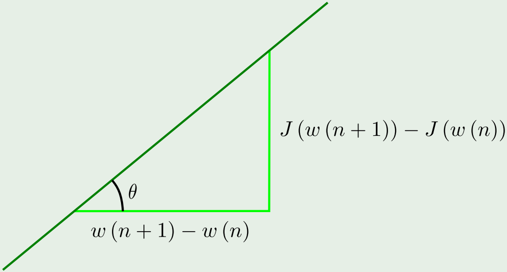
$$\tan\theta = \frac{J(w(n+1)) - J(w(n))}{w(n+1) - w(n)}$$

$$\tan\theta\,(w(n+1) - w(n)) = J(w(n+1)) - J(w(n))$$

$$J'(w(n))\,(w(n+1) - w(n)) = J(w(n+1)) - J(w(n))$$

Thus, we have that in $\mathbb{R}$

Remember Something quite Classic



$$
\begin{aligned}
\tan\theta &= \frac{J(w(n+1)) - J(w(n))}{w(n+1) - w(n)} \\
\tan\theta\,(w(n+1) - w(n)) &= J(w(n+1)) - J(w(n)) \\
J'(w(n))\,(w(n+1) - w(n)) &= J(w(n+1)) - J(w(n))
\end{aligned}
$$

Thus, we have that

Using the First Taylor expansion

$$J\left(w\left(n\right)\right) \approx J\left(w\left(n\right)\right) + J'\left(w\left(n\right)\right)\left[w\left(n+1\right) - w\left(n\right)\right] \quad (13)$$

Now, for Many Variables

An hyperplane in $\mathbb{R}^n$ is a set of the form

$$H = \left\{ \boldsymbol{x} | \boldsymbol{a}^T \boldsymbol{x} = b \right\} \tag{14}$$

Given $\boldsymbol{x} \in H$ and $\boldsymbol{x}_0 \in H$

$$b = \boldsymbol{a}^T \boldsymbol{x} = \boldsymbol{a}^T \boldsymbol{x}_0$$

Thus, we have that

$$H = \left\{ \boldsymbol{x} | \boldsymbol{a}^T (\boldsymbol{x} - \boldsymbol{x}_0) = 0 \right\}$$

Cinvestav

Now, for Many Variables

An hyperplane in $\mathbb{R}^n$ is a set of the form

$$H = \left\{ \boldsymbol{x} | \boldsymbol{a}^T \boldsymbol{x} = b \right\} \tag{14}$$

Given $\boldsymbol{x} \in H$ and $\boldsymbol{x}_0 \in H$

$$b = \boldsymbol{a}^T \boldsymbol{x} = \boldsymbol{a}^T \boldsymbol{x}_0$$

Thus, we have that

$$H = \left\{ \boldsymbol{x} | \boldsymbol{a}^T \left(\boldsymbol{x} - \boldsymbol{x}_0 \right) = 0 \right\}$$

Cinvestav

Now, for Many Variables

An hyperplane in $\mathbb{R}^n$ is a set of the form

$$H = \left\{ \boldsymbol{x} | \boldsymbol{a}^T \boldsymbol{x} = b \right\} \quad (14)$$

Given $\boldsymbol{x} \in H$ and $\boldsymbol{x}_0 \in H$

$$b = \boldsymbol{a}^T \boldsymbol{x} = \boldsymbol{a}^T \boldsymbol{x}_0$$

Thus, we have that

$$H = \left\{ \boldsymbol{x} | \boldsymbol{a}^T (\boldsymbol{x} - \boldsymbol{x}_0) = 0 \right\}$$

Thus, we have the following definition

Definition (Differentiability)

Assume that J is defined in a disk D containing $\boldsymbol{w}(n)$. We say that J is differentiable at $\boldsymbol{w}(n)$ if:

1. $\frac{\partial J(\boldsymbol{w}(n))}{\partial w_i}$ exist for all $i = 1, ..., n$.
2. J is locally linear at $\boldsymbol{w}(n)$.

Thus, we have the following definition

Definition (Differentiability)

Assume that J is defined in a disk D containing $\boldsymbol{w}(n)$. We say that J is differentiable at $\boldsymbol{w}(n)$ if:

1. $\frac{\partial J(\boldsymbol{w}(n))}{\partial w_i}$ exist for all $i = 1, ..., n$.
2. J is locally linear at $\boldsymbol{w}(n)$.

Thus, we have the following definition

Definition (Differentiability)

Assume that J is defined in a disk D containing $\boldsymbol{w}(n)$. We say that J is differentiable at $\boldsymbol{w}(n)$ if:

1. $\frac{\partial J(\boldsymbol{w}(n))}{\partial w_i}$ exist for all $i = 1, ..., n$.
2. J is locally linear at $\boldsymbol{w}(n)$.

Thus, given $J(\boldsymbol{w}(n))$

We know that we have the following operator

$$\nabla = \left(\frac{\partial}{\partial w_1}, \frac{\partial}{\partial w_2}, ..., \frac{\partial}{\partial w_m}\right) \quad (15)$$

Cinvestav

Thus, given $J\left(\boldsymbol{w}\left(n\right)\right)$

We know that we have the following operator

$$\nabla = \left(\frac{\partial}{\partial w_1}, \frac{\partial}{\partial w_2}, ..., \frac{\partial}{\partial w_m}\right) \tag{15}$$

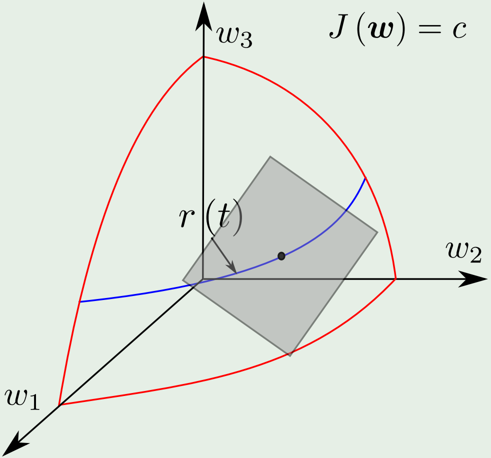
Thus, we have

$$\begin{aligned}\nabla J\left(\boldsymbol{w}\left(n\right)\right) &= \left(\frac{\partial J\left(\boldsymbol{w}\left(n\right)\right)}{\partial w_1}, \frac{\partial J\left(\boldsymbol{w}\left(n\right)\right)}{\partial w_2}, ..., \frac{\partial J\left(\boldsymbol{w}\left(n\right)\right)}{\partial w_m}\right)\\ &= \sum_{i=1}^{m} \hat{w}_i \frac{\partial J\left(\boldsymbol{w}\left(n\right)\right)}{\partial w_i}\end{aligned}$$

Where: $\hat{w}_i^T = (1, 0, ..., 0) \in \mathbb{R}$

Now

Given a curve function $r(t)$ that lies on the level set $J(\boldsymbol{w}(n)) = c$ (When is in $\mathbb{R}^3$)



Level Set

Definition

$$\{(w_1, w_2, ..., w_m) \in \mathbb{R}^m | J(w_1, w_2, ..., w_m) = c\} \quad (16)$$

Remark: In a normal Calculus course we will use x and f instead of $\boldsymbol{w}$ and J.

Where

Any curve has the following parametrization

$$r:[a,b]\to\mathbb{R}^m$$
$$r(t)=\left(w_1\left(t\right),...,w_m\left(t\right)\right)$$

With $r(n+1)=\left(w_1\left(n+1\right),...,w_m\left(n+1\right)\right)$

We can write the parametrized version of it

$$z(t)=J\left(w_1\left(t\right),w_2\left(t\right),...,w_m\left(t\right)\right)=c \quad (17)$$

Differentiating with respect to t and using the chain rule for multiple variables

$$\frac{dz(t)}{dt}=\sum_{i=1}^{m}\frac{\partial J\left(\boldsymbol{w}\left(t\right)\right)}{\partial w_i}\cdot\frac{dw_i(t)}{dt}=0 \quad (18)$$

Where

Any curve has the following parametrization

$$r : [a, b] \to \mathbb{R}^m$$

$$r(t) = (w_1(t), ..., w_m(t))$$

With $r(n+1) = (w_1(n+1), ..., w_m(n+1))$

We can write the parametrized version of it

$$z(t) = J(w_1(t), w_2(t), ..., w_m(t)) = c \quad (17)$$

Differentiating with respect to t and using the chain rule for multiple variables

$$\frac{dz(t)}{dt} = \sum_{i=1}^{m} \frac{\partial J(\boldsymbol{w}(t))}{\partial w_i} \cdot \frac{dw_i(t)}{dt} = 0 \quad (18)$$

Where

Any curve has the following parametrization

$$r : [a, b] \to \mathbb{R}^m$$
$$r(t) = (w_1(t), ..., w_m(t))$$

With $r(n+1) = (w_1(n+1), ..., w_m(n+1))$

We can write the parametrized version of it

$$z(t) = J(w_1(t), w_2(t), ..., w_m(t)) = c \quad (17)$$

Differentiating with respect to t and using the chain rule for multiple variables

$$\frac{dz(t)}{dt} = \sum_{i=1}^{m} \frac{\partial J(\boldsymbol{w}(t))}{\partial w_i} \cdot \frac{dw_i(t)}{dt} = 0 \quad (18)$$

Note

First

Given $y = f(\boldsymbol{u}) = (f_1(\boldsymbol{u}), ..., f_l(\boldsymbol{u}))$ and $\boldsymbol{u} = g(\boldsymbol{x}) = (g_1(\boldsymbol{x}), ..., g_m(\boldsymbol{x}))$.

We have then that

$$\frac{\partial(f_1, f_2, ..., f_l)}{\partial(x_1, x_2, ..., x_k)} = \frac{\partial(f_1, f_2, ..., f_l)}{\partial(g_1, g_2, ..., g_m)} \cdot \frac{\partial(g_1, g_2, ..., g_m)}{\partial(x_1, x_2, ..., x_k)} \quad (19)$$

Thus

$$\begin{aligned}\frac{\partial(f_1, f_2, ..., f_l)}{\partial x_i} &= \frac{\partial(f_1, f_2, ..., f_l)}{\partial(g_1, g_2, ..., g_m)} \cdot \frac{\partial(g_1, g_2, ..., g_m)}{\partial x_i} \\ &= \sum_{k=1}^{m} \frac{\partial(f_1, f_2, ..., f_l)}{\partial g_k} \frac{\partial g_k}{\partial x_i}\end{aligned}$$

Note

First

Given $y = f(\boldsymbol{u}) = (f_1(\boldsymbol{u}), ..., f_l(\boldsymbol{u}))$ and $\boldsymbol{u} = g(\boldsymbol{x}) = (g_1(\boldsymbol{x}), ..., g_m(\boldsymbol{x}))$.

We have then that

$$\frac{\partial(f_1, f_2, ..., f_l)}{\partial(x_1, x_2, ..., x_k)} = \frac{\partial(f_1, f_2, ..., f_l)}{\partial(g_1, g_2, ..., g_m)} \cdot \frac{\partial(g_1, g_2, ..., g_m)}{\partial(x_1, x_2, ..., x_k)} \tag{19}$$

Thus

$$\begin{aligned}\frac{\partial(f_1, f_2, ..., f_l)}{\partial x_i} &= \frac{\partial(f_1, f_2, ..., f_l)}{\partial(g_1, g_2, ..., g_m)} \cdot \frac{\partial(g_1, g_2, ..., g_m)}{\partial x_i} \\ &= \sum_{k=1}^{m} \frac{\partial(f_1, f_2, ..., f_l)}{\partial g_k} \frac{\partial g_k}{\partial x_i}\end{aligned}$$

Note

First

Given $y = f(\boldsymbol{u}) = (f_1(\boldsymbol{u}), ..., f_l(\boldsymbol{u}))$ and $\boldsymbol{u} = g(\boldsymbol{x}) = (g_1(\boldsymbol{x}), ..., g_m(\boldsymbol{x}))$.

We have then that

$$\frac{\partial(f_1, f_2, ..., f_l)}{\partial(x_1, x_2, ..., x_k)} = \frac{\partial(f_1, f_2, ..., f_l)}{\partial(g_1, g_2, ..., g_m)} \cdot \frac{\partial(g_1, g_2, ..., g_m)}{\partial(x_1, x_2, ..., x_k)} \tag{19}$$

Thus

$$\begin{aligned}\frac{\partial(f_1, f_2, ..., f_l)}{\partial x_i} &= \frac{\partial(f_1, f_2, ..., f_l)}{\partial(g_1, g_2, ..., g_m)} \cdot \frac{\partial(g_1, g_2, ..., g_m)}{\partial x_i} \\ &= \sum_{k=1}^{m} \frac{\partial(f_1, f_2, ..., f_l)}{\partial g_k} \frac{\partial g_k}{\partial x_i}\end{aligned}$$

Thus

Evaluating at $t = n$

$$\sum_{i=1}^{m} \frac{\partial J\left(\boldsymbol{w}\left(n\right)\right)}{\partial w_i} \cdot \frac{dw_i(n)}{dt} = 0$$

We have that

$$\nabla J\left(\boldsymbol{w}\left(n\right)\right) \cdot \boldsymbol{r}'\left(n\right) = 0 \tag{20}$$

This proves that for every level set the gradient is perpendicular to the tangent to any curve that lies on the level set

In particular to the point $\boldsymbol{w}\left(n\right)$.

Cinvestav

Thus

Evaluating at $t = n$

$$\sum_{i=1}^{m} \frac{\partial J\left(\boldsymbol{w}\left(n\right)\right)}{\partial w_i} \cdot \frac{dw_i(n)}{dt} = 0$$

We have that

$$\nabla J\left(\boldsymbol{w}\left(n\right)\right) \cdot r'\left(n\right) = 0 \tag{20}$$

This proves that for every level set the gradient is perpendicular to the tangent to any curve that lies on the level set

In particular to the point $w\left(n\right)$.

Thus

Evaluating at $t = n$

$$\sum_{i=1}^{m} \frac{\partial J\left(\boldsymbol{w}\left(n\right)\right)}{\partial w_i} \cdot \frac{dw_i(n)}{dt} = 0$$

We have that

$$\nabla J\left(\boldsymbol{w}\left(n\right)\right) \cdot r'\left(n\right) = 0 \tag{20}$$

This proves that for every level set the gradient is perpendicular to the tangent to any curve that lies on the level set

In particular to the point $\boldsymbol{w}\left(n\right)$.

Cinvestav

Now the tangent plane to the surface can be described generally

Thus

$$L(\boldsymbol{w}(n+1)) = J(\boldsymbol{w}(n)) + \nabla J^T(\boldsymbol{w}(\boldsymbol{n}))[\boldsymbol{w}(n+1) - \boldsymbol{w}(n)] \qquad (21)$$

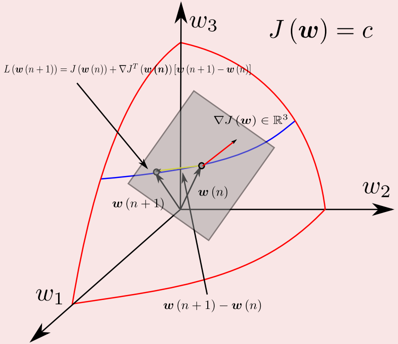
This looks like

Now the tangent plane to the surface can be described generally

Thus

$$L\left(\boldsymbol{w}\left(n+1\right)\right) = J\left(\boldsymbol{w}\left(n\right)\right) + \nabla J^{T}\left(\boldsymbol{w}\left(\boldsymbol{n}\right)\right)\left[\boldsymbol{w}\left(n+1\right) - \boldsymbol{w}\left(n\right)\right] \quad (21)$$

This looks like



Proving the fact about the Steepest Descent

We want the following

$$J\left(\boldsymbol{w}\left(n+1\right)\right) < J\left(\boldsymbol{w}\left(n\right)\right)$$

Using the first-order Taylor approximation

$$J\left(\boldsymbol{w}\left(n+1\right)\right) - J\left(\boldsymbol{w}\left(n\right)\right) \approx \nabla J^{T}\left(\boldsymbol{w}\left(n\right)\right)\Delta\boldsymbol{w}\left(n\right)$$

So, we ask the following

$$\Delta\boldsymbol{w}\left(n\right) \approx -\eta\nabla J\left(\boldsymbol{w}\left(n\right)\right) \text{ with } \eta > 0$$

Proving the fact about the Steepest Descent

We want the following

$$J\left(\boldsymbol{w}\left(n+1\right)\right) < J\left(\boldsymbol{w}\left(n\right)\right)$$

Using the first-order Taylor approximation

$$J\left(\boldsymbol{w}\left(n+1\right)\right) - J\left(\boldsymbol{w}\left(n\right)\right) \approx \nabla J^{T}\left(\boldsymbol{w}\left(\boldsymbol{n}\right)\right)\Delta\boldsymbol{w}\left(n\right)$$

So, we ask the following

$$\Delta\boldsymbol{w}\left(n\right) \approx -\eta\nabla J\left(\boldsymbol{w}\left(\boldsymbol{n}\right)\right) \text{ with } \eta > 0$$

Proving the fact about the Steepest Descent

We want the following

$$J\left(\boldsymbol{w}\left(n+1\right)\right) < J\left(\boldsymbol{w}\left(n\right)\right)$$

Using the first-order Taylor approximation

$$J\left(\boldsymbol{w}\left(n+1\right)\right) - J\left(\boldsymbol{w}\left(n\right)\right) \approx \nabla J^{T}\left(\boldsymbol{w}\left(\boldsymbol{n}\right)\right)\Delta\boldsymbol{w}\left(n\right)$$

So, we ask the following

$$\Delta\boldsymbol{w}\left(n\right) \approx -\eta\nabla J\left(\boldsymbol{w}\left(\boldsymbol{n}\right)\right) \text{ with } \eta > 0$$

Then

We have that

$$J\left(\boldsymbol{w}\left(n+1\right)\right)-J\left(\boldsymbol{w}\left(n\right)\right)\approx-\eta\nabla J^{T}\left(\boldsymbol{w}\left(\boldsymbol{n}\right)\right)\nabla J\left(\boldsymbol{w}\left(\boldsymbol{n}\right)\right)=-\eta\left\Vert \nabla J\left(\boldsymbol{w}\left(\boldsymbol{n}\right)\right)\right\Vert ^{2}$$

Thus

$$J\left(\boldsymbol{w}\left(n+1\right)\right)-J\left(\boldsymbol{w}\left(n\right)\right)<0$$

Or

$$J\left(\boldsymbol{w}\left(n+1\right)\right)<J\left(\boldsymbol{w}\left(n\right)\right)$$

Cinvestav

Then

We have that

$$J\left(\boldsymbol{w}\left(n+1\right)\right)-J\left(\boldsymbol{w}\left(n\right)\right)\approx-\eta\nabla J^{T}\left(\boldsymbol{w}\left(\boldsymbol{n}\right)\right)\nabla J\left(\boldsymbol{w}\left(\boldsymbol{n}\right)\right)=-\eta\left\Vert \nabla J\left(\boldsymbol{w}\left(\boldsymbol{n}\right)\right)\right\Vert ^{2}$$

Thus

$$J\left(\boldsymbol{w}\left(n+1\right)\right)-J\left(\boldsymbol{w}\left(n\right)\right)<0$$

Or

$$J\left(\boldsymbol{w}\left(n+1\right)\right)<J\left(\boldsymbol{w}\left(n\right)\right)$$

Cinvestav

Then

We have that

$$J\left(\boldsymbol{w}\left(n+1\right)\right)-J\left(\boldsymbol{w}\left(n\right)\right)\approx-\eta\nabla J^{T}\left(\boldsymbol{w}\left(\boldsymbol{n}\right)\right)\nabla J\left(\boldsymbol{w}\left(\boldsymbol{n}\right)\right)=-\eta\left\Vert \nabla J\left(\boldsymbol{w}\left(\boldsymbol{n}\right)\right)\right\Vert ^{2}$$

Thus

$$J\left(\boldsymbol{w}\left(n+1\right)\right)-J\left(\boldsymbol{w}\left(n\right)\right)<0$$

Or

$$J\left(\boldsymbol{w}\left(n+1\right)\right)<J\left(\boldsymbol{w}\left(n\right)\right)$$

Outline

Newton's Method

Here

The basic idea of Newton's method is to minimize the quadratic approximation of the cost function $J(\boldsymbol{w})$ around the current point $\boldsymbol{w}(n)$.

Using a second-order Taylor series expansion of the cost function around the point $\boldsymbol{w}(n)$

$$\begin{aligned}\Delta J(\boldsymbol{w}(n)) &= J(\boldsymbol{w}(n+1)) - J(\boldsymbol{w}(n)) \\ &\approx \nabla J^T(\boldsymbol{w}(\boldsymbol{n}))\Delta\boldsymbol{w}(n) + \frac{1}{2}\Delta\boldsymbol{w}^T(n)H(n)\Delta\boldsymbol{w}(n)\end{aligned}$$

Where given that $\boldsymbol{w}(n)$ is a vector with dimension m

$$H = \nabla^2 J(\boldsymbol{w}) = \begin{pmatrix} \frac{\partial^2 J(\boldsymbol{w})}{\partial w_1^2} & \frac{\partial^2 J(\boldsymbol{w})}{\partial w_1 \partial w_2} & \cdots & \frac{\partial^2 J(\boldsymbol{w})}{\partial w_1 \partial w_m} \\ \frac{\partial^2 J(\boldsymbol{w})}{\partial w_2 \partial w_1} & \frac{\partial^2 J(\boldsymbol{w})}{\partial w_2^2} & \cdots & \frac{\partial^2 J(\boldsymbol{w})}{\partial w_2 \partial w_m} \\ \vdots & \vdots & & \vdots \\ \frac{\partial^2 J(\boldsymbol{w})}{\partial w_m \partial w_1} & \frac{\partial^2 J(\boldsymbol{w})}{\partial w_m \partial w_2} & \cdots & \frac{\partial^2 J(\boldsymbol{w})}{\partial w_m^2} \end{pmatrix}$$

Newton's Method

Here

The basic idea of Newton's method is to minimize the quadratic approximation of the cost function $J(\boldsymbol{w})$ around the current point $\boldsymbol{w}(n)$.

Using a second-order Taylor series expansion of the cost function around the point $\boldsymbol{w}(n)$

$$
\begin{aligned}
\Delta J(\boldsymbol{w}(n)) &= J(\boldsymbol{w}(n+1)) - J(\boldsymbol{w}(n)) \\
&\approx \nabla J^T(\boldsymbol{w}(\boldsymbol{n}))\,\Delta\boldsymbol{w}(n) + \frac{1}{2}\Delta\boldsymbol{w}^T(n)\,H(n)\,\Delta\boldsymbol{w}(n)
\end{aligned}
$$

Where given that $w(n)$ is a vector with dimension m

$$
H = \nabla^2 J(\boldsymbol{w}) = \begin{pmatrix}
\frac{\partial^2 J(\boldsymbol{w})}{\partial w_1^2} & \frac{\partial^2 J(\boldsymbol{w})}{\partial w_1 \partial w_2} & \cdots & \frac{\partial^2 J(\boldsymbol{w})}{\partial w_1 \partial w_m} \\
\frac{\partial^2 J(\boldsymbol{w})}{\partial w_2 \partial w_1} & \frac{\partial^2 J(\boldsymbol{w})}{\partial w_2^2} & \cdots & \frac{\partial^2 J(\boldsymbol{w})}{\partial w_2 \partial w_m} \\
\vdots & \vdots & & \vdots \\
\frac{\partial^2 J(\boldsymbol{w})}{\partial w_m \partial w_1} & \frac{\partial^2 J(\boldsymbol{w})}{\partial w_m \partial w_2} & \cdots & \frac{\partial^2 J(\boldsymbol{w})}{\partial w_m^2}
\end{pmatrix}
$$

Newton's Method

Here

The basic idea of Newton's method is to minimize the quadratic approximation of the cost function $J(\boldsymbol{w})$ around the current point $\boldsymbol{w}(n)$.

Using a second-order Taylor series expansion of the cost function around the point $\boldsymbol{w}(n)$

$$\begin{aligned}\Delta J(\boldsymbol{w}(n)) &= J(\boldsymbol{w}(n+1)) - J(\boldsymbol{w}(n)) \\ &\approx \nabla J^T(\boldsymbol{w}(\boldsymbol{n}))\,\Delta\boldsymbol{w}(n) + \frac{1}{2}\Delta\boldsymbol{w}^T(n)\,H(n)\,\Delta\boldsymbol{w}(n)\end{aligned}$$

Where given that $\boldsymbol{w}(n)$ is a vector with dimension m

$$H = \nabla^2 J(\boldsymbol{w}) = \begin{pmatrix} \frac{\partial^2 J(\boldsymbol{w})}{\partial w_1^2} & \frac{\partial^2 J(\boldsymbol{w})}{\partial w_1 \partial w_2} & \cdots & \frac{\partial^2 J(\boldsymbol{w})}{\partial w_1 \partial w_m} \\ \frac{\partial^2 J(\boldsymbol{w})}{\partial w_2 \partial w_1} & \frac{\partial^2 J(\boldsymbol{w})}{\partial w_2^2} & \cdots & \frac{\partial^2 J(\boldsymbol{w})}{\partial w_2 \partial w_m} \\ \vdots & \vdots & & \vdots \\ \frac{\partial^2 J(\boldsymbol{w})}{\partial w_m \partial w_1} & \frac{\partial^2 J(\boldsymbol{w})}{\partial w_m \partial w_2} & \cdots & \frac{\partial^2 J(\boldsymbol{w})}{\partial w_m^2} \end{pmatrix}$$

Now, we want to minimize $J\left(\boldsymbol{w}\left(n+1\right)\right)$

Do you have any idea?

Look again

$$J\left(\boldsymbol{w}\left(n\right)\right)+\nabla J^{T}\left(\boldsymbol{w}\left(\boldsymbol{n}\right)\right)\Delta\boldsymbol{w}\left(n\right)+\frac{1}{2}\Delta\boldsymbol{w}^{T}\left(n\right)H\left(n\right)\Delta\boldsymbol{w}\left(n\right) \quad (22)$$

Derive with respect to $\Delta\boldsymbol{w}\left(n\right)$

$$\nabla J\left(\boldsymbol{w}\left(\boldsymbol{n}\right)\right)+H\left(n\right)\Delta\boldsymbol{w}\left(n\right)=0 \quad (23)$$

Thus

$$\Delta\boldsymbol{w}\left(n\right)=-H^{-1}\left(n\right)\nabla J\left(\boldsymbol{w}\left(\boldsymbol{n}\right)\right)$$

Cinvestav

Now, we want to minimize $J\left(\boldsymbol{w}\left(n+1\right)\right)$

Do you have any idea?

Look again

$$J\left(\boldsymbol{w}\left(n\right)\right)+\nabla J^{T}\left(\boldsymbol{w}\left(\boldsymbol{n}\right)\right)\Delta\boldsymbol{w}\left(n\right)+\frac{1}{2}\Delta\boldsymbol{w}^{T}\left(n\right)H\left(n\right)\Delta\boldsymbol{w}\left(n\right) \quad (22)$$

Derive with respect to $\Delta\boldsymbol{w}\left(n\right)$

$$\nabla J\left(\boldsymbol{w}\left(\boldsymbol{n}\right)\right)+H\left(n\right)\Delta\boldsymbol{w}\left(n\right)=0 \quad (23)$$

Thus

$$\Delta\boldsymbol{w}\left(n\right)=-H^{-1}\left(n\right)\nabla J\left(\boldsymbol{w}\left(\boldsymbol{n}\right)\right)$$

Now, we want to minimize $J(\boldsymbol{w}(n+1))$

Do you have any idea?

Look again

$$J(\boldsymbol{w}(n)) + \nabla J^T(\boldsymbol{w}(\boldsymbol{n}))\Delta\boldsymbol{w}(n) + \frac{1}{2}\Delta\boldsymbol{w}^T(n)H(n)\Delta\boldsymbol{w}(n) \quad (22)$$

Derive with respect to $\Delta\boldsymbol{w}(n)$

$$\nabla J(\boldsymbol{w}(\boldsymbol{n})) + H(n)\Delta\boldsymbol{w}(n) = 0 \quad (23)$$

Thus

$$\Delta\boldsymbol{w}(n) = -H^{-1}(n)\nabla J(\boldsymbol{w}(\boldsymbol{n}))$$

Cinvestav

The Final Method

Define the following

$$J\left(\boldsymbol{w}\left(n+1\right)\right)-J\left(\boldsymbol{w}\left(n\right)\right)=-H^{-1}\left(n\right)\nabla J\left(\boldsymbol{w}\left(\boldsymbol{n}\right)\right)$$

Then

$$J\left(\boldsymbol{w}\left(n+1\right)\right)=J\left(\boldsymbol{w}\left(n\right)\right)-H^{-1}\left(n\right)\nabla J\left(\boldsymbol{w}\left(\boldsymbol{n}\right)\right)$$

The Final Method

Define the following

$$J\left(\boldsymbol{w}\left(n+1\right)\right)-J\left(\boldsymbol{w}\left(n\right)\right)=-H^{-1}\left(n\right)\nabla J\left(\boldsymbol{w}\left(\boldsymbol{n}\right)\right)$$

Then

$$J\left(\boldsymbol{w}\left(n+1\right)\right)=J\left(\boldsymbol{w}\left(n\right)\right)-H^{-1}\left(n\right)\nabla J\left(\boldsymbol{w}\left(\boldsymbol{n}\right)\right)$$

Cinvestav

Outline

Cinvestav

We have then an error

Something Notable

$$J(\boldsymbol{w}) = \frac{1}{2}\sum_{i=1}^{n} e^2(i)$$

Thus using the first order Taylor expansion, we linearize

$$e_l(i, \boldsymbol{w}) = e(i) + \left[\frac{\partial e(i)}{\partial \boldsymbol{w}}\right]^T [\boldsymbol{w} - \boldsymbol{w}(n)]$$

In matrix form

$$\boldsymbol{e}_l(n, \boldsymbol{w}) = \boldsymbol{e}(n) + \mathbf{J}(n)[\boldsymbol{w} - \boldsymbol{w}(n)]$$

We have then an error

Something Notable

$$J(\boldsymbol{w}) = \frac{1}{2}\sum_{i=1}^{n} e^2(i)$$

Thus using the first order Taylor expansion, we linearize

$$e_l(i, \boldsymbol{w}) = e(i) + \left[\frac{\partial e(i)}{\partial \boldsymbol{w}}\right]^T [\boldsymbol{w} - \boldsymbol{w}(n)]$$

In matrix form

$$\boldsymbol{e}_l(n, \boldsymbol{w}) = \boldsymbol{e}(n) + \mathbf{J}(n)[\boldsymbol{w} - \boldsymbol{w}(n)]$$

We have then an error

Something Notable

$$J(\boldsymbol{w}) = \frac{1}{2}\sum_{i=1}^{n} e^2(i)$$

Thus using the first order Taylor expansion, we linearize

$$e_l(i, \boldsymbol{w}) = e(i) + \left[\frac{\partial e(i)}{\partial \boldsymbol{w}}\right]^T [\boldsymbol{w} - \boldsymbol{w}(n)]$$

In matrix form

$$\boldsymbol{e}_l(n, \boldsymbol{w}) = \boldsymbol{e}(n) + \mathsf{J}(n)[\boldsymbol{w} - \boldsymbol{w}(n)]$$

Where

The error vector is equal to

$$\boldsymbol{e}(n) = [e(1), e(2), ..., e(n)]^T \tag{24}$$

Thus, we get the famous Jacobian once we derive $\frac{\partial e(i)}{\partial w}$

$$J(n) = \begin{pmatrix} \frac{\partial e(1)}{\partial w_1} & \frac{\partial e(1)}{\partial w_2} & \cdots & \frac{\partial e(1)}{\partial w_m} \\ \frac{\partial e(2)}{\partial w_1} & \frac{\partial e(2)}{\partial w_2} & \cdots & \frac{\partial e(2)}{\partial w_m} \\ \vdots & \vdots & & \vdots \\ \frac{\partial e(n)}{\partial w_1} & \frac{\partial e(n)}{\partial w_2} & \cdots & \frac{\partial e(n)}{\partial w_m} \end{pmatrix}$$

Where

The error vector is equal to

$$\boldsymbol{e}(n) = [e(1), e(2), ..., e(n)]^T \tag{24}$$

Thus, we get the famous Jacobian once we derive $\frac{\partial e(i)}{\partial \boldsymbol{w}}$

$$\mathsf{J}(n) = \begin{pmatrix} \frac{\partial e(1)}{\partial w_1} & \frac{\partial e(1)}{\partial w_2} & \cdots & \frac{\partial e(1)}{\partial w_m} \\ \frac{\partial e(2)}{\partial w_1} & \frac{\partial e(2)}{\partial w_2} & \cdots & \frac{\partial e(2)}{\partial w_m} \\ \vdots & \vdots & & \vdots \\ \frac{\partial e(n)}{\partial w_1} & \frac{\partial e(n)}{\partial w_2} & \cdots & \frac{\partial e(n)}{\partial w_m} \end{pmatrix}$$

Where

We want the following

$$\boldsymbol{w}(n+1) = arg\min_{\boldsymbol{w}} \left\{ \frac{1}{2} \left\| \boldsymbol{e}_l(n, \boldsymbol{w}) \right\|^2 \right\}$$

Ideas

What if we expand out the equation?

Where

We want the following

$$\boldsymbol{w}(n+1) = arg \min_{\boldsymbol{w}} \left\{ \frac{1}{2} \left\| \boldsymbol{e}_l(n, \boldsymbol{w}) \right\|^2 \right\}$$

Ideas

What if we expand out the equation?

Expanded Version

We get

$$\frac{1}{2}\left\|\boldsymbol{e}_l\left(n,\boldsymbol{w}\right)\right\|^2 = \frac{1}{2}\left\|\boldsymbol{e}\left(n\right)\right\|^2 + \boldsymbol{e}^T\left(n\right)\boldsymbol{J}\left(n\right)\left(\boldsymbol{w}-\boldsymbol{w}\left(n\right)\right) + \ldots$$
$$\frac{1}{2}\left(\boldsymbol{w}-\boldsymbol{w}\left(n\right)\right)^T\boldsymbol{J}^T\left(n\right)\boldsymbol{J}\left(n\right)\left(\boldsymbol{w}-\boldsymbol{w}\left(n\right)\right)$$

Cinvestav

Now,doing the Differential, we get

Differentiating the equation with respect to $\boldsymbol{w}$

$$\mathsf{J}^{T}(n)\,\boldsymbol{e}(n) + \mathsf{J}^{T}(n)\,\mathsf{J}(n)\left[\boldsymbol{w} - \boldsymbol{w}(n)\right] = 0$$

We get finally

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) - \left(\mathsf{J}^{T}(n)\,\mathsf{J}(n)\right)^{-1}\mathsf{J}^{T}(n)\,\boldsymbol{e}(n) \qquad (25)$$

Now,doing the Differential, we get

Differentiating the equation with respect to $\boldsymbol{w}$

$$\mathsf{J}^{T}(n)\,\boldsymbol{e}(n)+\mathsf{J}^{T}(n)\,\mathsf{J}(n)\left[\boldsymbol{w}-\boldsymbol{w}(n)\right]=0$$

We get finally

$$\boldsymbol{w}(n+1)=\boldsymbol{w}(n)-\left(\mathsf{J}^{T}(n)\,\mathsf{J}(n)\right)^{-1}\mathsf{J}^{T}(n)\,e(n) \quad (25)$$

Cinvestav

Remarks

We have that

- The Newton's method that requires knowledge of the Hessian matrix of the cost function.
- The Gauss-Newton method only requires the Jacobian matrix of the error vector $\boldsymbol{e}(n)$.

However

The Gauss-Newton iteration to be computable, the matrix product $\mathbf{J}^T(n)\,\mathbf{J}(n)$ must be nonsingular!!!

Remarks

We have that

- The Newton's method that requires knowledge of the Hessian matrix of the cost function.
- The Gauss-Newton method only requires the Jacobian matrix of the error vector $\boldsymbol{e}(n)$.

However

The Gauss-Newton iteration to be computable, the matrix product $\mathsf{J}^T(n)\,\mathsf{J}(n)$ must be nonsingular!!!

Outline

Cinvestav

Introduction

A linear least-squares filter has two distinctive characteristics

- First, the single neuron around which it is built is linear.
- The cost function $J(\boldsymbol{w})$ used to design the filter consists of the sum of error squares.

Thus, expressing the error

$$e(n) = d(n) - (\boldsymbol{x}(1), \ldots, \boldsymbol{x}(n))^T \boldsymbol{w}(n)$$

Short Version - error is linear in the weight vector $\boldsymbol{w}(n)$

$$e(n) = d(n) - \boldsymbol{X}(n)\boldsymbol{w}(n)$$

- Where $d(n)$ is a $n \times 1$ desired response vector.
- Where $\boldsymbol{X}(n)$ is the $n \times m$ data matrix.

Introduction

A linear least-squares filter has two distinctive characteristics

- First, the single neuron around which it is built is linear.
- The cost function $J(\boldsymbol{w})$ used to design the filter consists of the sum of error squares.

Thus, expressing the error

$$e(n) = d(n) - (\boldsymbol{x}(1), ..., \boldsymbol{x}(n))^T \boldsymbol{w}(n)$$

Short Version - error is linear in the weight vector $\boldsymbol{w}(n)$

$$e(n) = d(n) - \boldsymbol{X}(n)\boldsymbol{w}(n)$$

- Where $d(n)$ is a $n \times 1$ desired response vector.
- Where $\boldsymbol{X}(n)$ is the $n \times m$ data matrix.

Introduction

A linear least-squares filter has two distinctive characteristics

- First, the single neuron around which it is built is linear.
- The cost function $J(\boldsymbol{w})$ used to design the filter consists of the sum of error squares.

Thus, expressing the error

$$e(n) = d(n) - (\boldsymbol{x}(1), ..., \boldsymbol{x}(n))^T \boldsymbol{w}(n)$$

Short Version - error is linear in the weight vector $\boldsymbol{w}(n)$

$$e(n) = d(n) - \boldsymbol{X}(n)\boldsymbol{w}(n)$$

- Where $d(n)$ is a $n \times 1$ desired response vector.
- Where $\boldsymbol{X}(n)$ is the $n \times m$ data matrix.

Now, differentiate $e(n)$ with respect to $\boldsymbol{w}(n)$

Thus

$$\nabla e(n) = -\boldsymbol{X}^T(n)$$

Correspondingly, the Jacobian of $e(n)$ is

$$\mathsf{J}(n) = -\boldsymbol{X}(n)$$

Let us to use the Gaussian-Newton

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) - \left(\mathsf{J}^T(n)\,\mathsf{J}(n)\right)^{-1}\mathsf{J}^T(n)\,e(n)$$

Cinvestav

Now, differentiate $e(n)$ with respect to $\boldsymbol{w}(n)$

Thus

$$\nabla e(n) = -\boldsymbol{X}^T(n)$$

Correspondingly, the Jacobian of $e(n)$ is

$$\mathsf{J}(n) = -\boldsymbol{X}(n)$$

Let us to use the Gaussian-Newton

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) - \left(\mathsf{J}^T(n)\,\mathsf{J}(n)\right)^{-1}\mathsf{J}^T(n)\,e(n)$$

Now, differentiate $e(n)$ with respect to $\boldsymbol{w}(n)$

Thus

$$\nabla e(n) = -\boldsymbol{X}^{T}(n)$$

Correspondingly, the Jacobian of $e(n)$ is

$$\mathsf{J}(n) = -\boldsymbol{X}(n)$$

Let us to use the Gaussian-Newton

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) - \left(\mathsf{J}^{T}(n)\,\mathsf{J}(n)\right)^{-1}\mathsf{J}^{T}(n)\,e(n)$$

Cinvestav

Thus

We have the following

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) - \left(-\boldsymbol{X}^T(n) \cdot -\boldsymbol{X}(n)\right)^{-1} \cdot -\boldsymbol{X}^T(n)\left[d(n) - \boldsymbol{X}(n)\boldsymbol{w}(n)\right]$$

We have then

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \left(\boldsymbol{X}^T(n)\boldsymbol{X}(n)\right)^{-1}\boldsymbol{X}^T(n)d(n) - \dots$$
$$\left(\boldsymbol{X}^T(n)\boldsymbol{X}(n)\right)^{-1}\boldsymbol{X}^T(n)\boldsymbol{X}(n)\boldsymbol{w}(n)$$

Thus, we have

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \left(\boldsymbol{X}^T(n)\boldsymbol{X}(n)\right)^{-1}\boldsymbol{X}^T(n)d(n) - \boldsymbol{w}(n)$$
$$= \left(\boldsymbol{X}^T(n)\boldsymbol{X}(n)\right)^{-1}\boldsymbol{X}^T(n)d(n)$$

Thus

We have the following

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) - \left(-\boldsymbol{X}^T(n) \cdot -\boldsymbol{X}(n)\right)^{-1} \cdot -\boldsymbol{X}^T(n)\left[d(n) - \boldsymbol{X}(n)\boldsymbol{w}(n)\right]$$

We have then

$$\begin{aligned}\boldsymbol{w}(n+1) =& \boldsymbol{w}(n) + \left(\boldsymbol{X}^T(n)\boldsymbol{X}(n)\right)^{-1}\boldsymbol{X}^T(n)\, d(n) - \dots \\ & \left(\boldsymbol{X}^T(n)\boldsymbol{X}(n)\right)^{-1}\boldsymbol{X}^T(n)\boldsymbol{X}(n)\boldsymbol{w}(n)\end{aligned}$$

Thus, we have

$$\begin{aligned}\boldsymbol{w}(n+1) =& \boldsymbol{w}(n) + \left(\boldsymbol{X}^T(n)\boldsymbol{X}(n)\right)^{-1}\boldsymbol{X}^T(n)\, d(n) - \boldsymbol{w}(n) \\ =& \left(\boldsymbol{X}^T(n)\boldsymbol{X}(n)\right)^{-1}\boldsymbol{X}^T(n)\, d(n)\end{aligned}$$

Thus

We have the following

$$\boldsymbol{w}(n+1)=\boldsymbol{w}(n)-\left(-\boldsymbol{X}^{T}(n)\cdot-\boldsymbol{X}(n)\right)^{-1}\cdot-\boldsymbol{X}^{T}(n)\left[d(n)-\boldsymbol{X}(n)\boldsymbol{w}(n)\right]$$

We have then

$$\begin{aligned}\boldsymbol{w}(n+1)=&\boldsymbol{w}(n)+\left(\boldsymbol{X}^{T}(n)\boldsymbol{X}(n)\right)^{-1}\boldsymbol{X}^{T}(n)d(n)-...\\&\left(\boldsymbol{X}^{T}(n)\boldsymbol{X}(n)\right)^{-1}\boldsymbol{X}^{T}(n)\boldsymbol{X}(n)\boldsymbol{w}(n)\end{aligned}$$

Thus, we have

$$\begin{aligned}\boldsymbol{w}(\boldsymbol{n}+\mathbf{1})=&\boldsymbol{w}(n)+\left(\boldsymbol{X}^{T}(n)\boldsymbol{X}(n)\right)^{-1}\boldsymbol{X}^{T}(n)d(n)-\boldsymbol{w}(n)\\=&\left(\boldsymbol{X}^{T}(n)\boldsymbol{X}(n)\right)^{-1}\boldsymbol{X}^{T}(n)d(n)\end{aligned}$$

Outline

Cinvestav

Again Our Error Cost function

We have

$$J(\boldsymbol{w}) = \frac{1}{2} e^2(n)$$

where $e(n)$ is the error signal measured at time n.

Again differentiating against the vector $\boldsymbol{w}$

$$\frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}} = e(n) \frac{\partial e(n)}{\partial \boldsymbol{w}}$$

LMS algorithm operates with a linear neuron so we may express the error signal as

$$e(n) = d(n) - \boldsymbol{x}^T(n)\boldsymbol{w}(n) \quad (26)$$

Again Our Error Cost function

We have

$$J(\boldsymbol{w}) = \frac{1}{2} e^2(n)$$

where $e(n)$ is the error signal measured at time n.

Again differentiating against the vector $\boldsymbol{w}$

$$\frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}} = e(n) \frac{\partial e(n)}{\partial \boldsymbol{w}}$$

LMS algorithm operates with a linear neuron so we may express the error signal as

$$e(n) = d(n) - \boldsymbol{x}^T(n) \boldsymbol{w}(n) \quad (26)$$

Again Our Error Cost function

We have

$$J(\boldsymbol{w}) = \frac{1}{2} e^2(n)$$

where $e(n)$ is the error signal measured at time n.

Again differentiating against the vector $\boldsymbol{w}$

$$\frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}} = e(n) \frac{\partial e(n)}{\partial \boldsymbol{w}}$$

LMS algorithm operates with a linear neuron so we may express the error signal as

$$e(n) = d(n) - \boldsymbol{x}^T(n)\, \boldsymbol{w}(n) \tag{26}$$

Cinvestav

We have

Something Notable

$$\frac{\partial e(n)}{\partial \boldsymbol{w}} = -\boldsymbol{x}(n)$$

Then

$$\frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}} = -\boldsymbol{x}(n)\, e(n)$$

Using this as an estimate for the gradient vector, we have for the gradient descent

$$\widehat{\boldsymbol{w}}(n+1) = \widehat{\boldsymbol{w}}(n) + \eta \boldsymbol{x}(n)\, e(n) \tag{27}$$

We have

Something Notable

$$\frac{\partial e(n)}{\partial \boldsymbol{w}} = -\boldsymbol{x}(n)$$

Then

$$\frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}} = -\boldsymbol{x}(n)\, e(n)$$

Using this as an estimate for the gradient vector, we have for the gradient descent

$$\widehat{\boldsymbol{w}}(n+1) = \widehat{\boldsymbol{w}}(n) + \eta \boldsymbol{x}(n)\, e(n) \quad (27)$$

We have

Something Notable

$$\frac{\partial e(n)}{\partial \boldsymbol{w}} = -\boldsymbol{x}(n)$$

Then

$$\frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}} = -\boldsymbol{x}(n)\, e(n)$$

Using this as an estimate for the gradient vector, we have for the gradient descent

$$\widehat{\boldsymbol{w}}(n+1) = \widehat{\boldsymbol{w}}(n) + \eta \boldsymbol{x}(n)\, e(n) \tag{27}$$

Cinvestav

Remarks

The feedback loop around the weight vector low-pass filter

- It behaves like a low-pass filter.
- It passes the low frequency component of the error signal and attenuating its high frequency component.

Cinvestav

Remarks

The feedback loop around the weight vector low-pass filter

- It behaves like a low-pass filter.
- It passes the low frequency component of the error signal and attenuating its high frequency component.

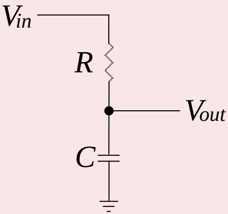
Low-Pass filter

Remarks

The feedback loop around the weight vector low-pass filter

- It behaves like a low-pass filter.
- It passes the low frequency component of the error signal and attenuating its high frequency component.

Low-Pass filter



Actually

Thus

The average time constant of this filtering action is inversely proportional to the learning-rate parameter η.

Actually

Thus

The average time constant of this filtering action is inversely proportional to the learning-rate parameter η.

Thus

Assigning a small value to η, the adaptive process progresses slowly.

- Thus, LMS is a more accurate filter.

Actually

Thus

The average time constant of this filtering action is inversely proportional to the learning-rate parameter η.

Thus

Assigning a small value to η, the adaptive process progresses slowly.

Thus

- More of the past data are remembered by the LMS algorithm.
- Thus, LMS is a more accurate filter.

Cinvestav

Actually

Thus

The average time constant of this filtering action is inversely proportional to the learning-rate parameter η.

Thus

Assigning a small value to η, the adaptive process progresses slowly.

Thus

- More of the past data are remembered by the LMS algorithm.
- Thus, LMS is a more accurate filter.

Outline

Cinvestav

Convergence of the LMS

This convergence depends on the following points

- The statistical characteristics of the input vector $\boldsymbol{x}(n)$.
- The learning-rate parameter η.

Something Notable

However instead using $E[\hat{\boldsymbol{w}}(n)]$ as $n \to \infty$, we use $E[e^2(n)] \to$constant as $n \to \infty$

Convergence of the LMS

This convergence depends on the following points

- The statistical characteristics of the input vector $\boldsymbol{x}(n)$.
- The learning-rate parameter η.

Something Notable

However instead using $E\left[\widehat{\boldsymbol{w}}(n)\right]$ as $n \to \infty$, we use $E\left[e^2(n)\right] \to$constant as $n \to \infty$

To make this analysis practical

We take the following assumptions

- The successive input vectors $\boldsymbol{x}(1), \boldsymbol{x}(2), ..$ are statistically independent of each other.
- At time step n, the input vector $\boldsymbol{x}(n)$ is statistically independent of all previous samples of the desired response, namely $d(1), d(2), ..., d(n-1)$.
- At time step n, the desired response $d(n)$ is dependent on $\boldsymbol{x}(n)$, but statistically independent of all previous values of the desired response.
- The input vector $\boldsymbol{x}(n)$ and desired response $d(n)$ are drawn from Gaussiandistributed populations.

To make this analysis practical

We take the following assumptions

- The successive input vectors $\boldsymbol{x}(1), \boldsymbol{x}(2), ..$ are statistically independent of each other.
- At time step n, the input vector $\boldsymbol{x}(n)$ is statistically independent of all previous samples of the desired response, namely $d(1), d(2), ..., d(n-1)$.
- At time step n, the desired response $d(n)$ is dependent on $x(n)$, but statistically independent of all previous values of the desired response.
- The input vector $x(n)$ and desired response $d(n)$ are drawn from Gaussiandistributed populations.

Cinvestav

To make this analysis practical

We take the following assumptions

- The successive input vectors $\boldsymbol{x}(1), \boldsymbol{x}(2), ..$ are statistically independent of each other.
- At time step n, the input vector $\boldsymbol{x}(n)$ is statistically independent of all previous samples of the desired response, namely $d(1), d(2), ..., d(n-1)$.
- At time step n, the desired response $d(n)$ is dependent on $x(n)$, but statistically independent of all previous values of the desired response.
- The input vector $x(n)$ and desired response $d(n)$ are drawn from Gaussiandistributed populations.

To make this analysis practical

We take the following assumptions

- The successive input vectors $\boldsymbol{x}(1), \boldsymbol{x}(2), ..$ are statistically independent of each other.
- At time step n, the input vector $\boldsymbol{x}(n)$ is statistically independent of all previous samples of the desired response, namely $d(1), d(2), ..., d(n-1)$.
- At time step n, the desired response $d(n)$ is dependent on $x(n)$, but statistically independent of all previous values of the desired response.
- The input vector $x(n)$ and desired response $d(n)$ are drawn from Gaussiandistributed populations.

We get the following

The LMS is convergent in the mean square provided that η satisfies

$$0 < \eta < \frac{2}{\lambda_{\max}} \tag{28}$$

Because $\lambda_{\max}$ is the largest eigenvalue of the correlation sample $\boldsymbol{R}_x$

This can be difficult in reality.... then we use the trace instead

$$0 < \eta < \frac{2}{\text{trace}\left[\boldsymbol{R}_x\right]} \tag{29}$$

However, each diagonal element of $\boldsymbol{R}_x$ is equal the mean-squared value of the corresponding of the sensor input

We can re-state the previous condition as

$$0 < \eta < \frac{2}{\text{sum of the mean-square values of the sensor input}} \tag{30}$$

We get the following

The LMS is convergent in the mean square provided that η satisfies

$$0 < \eta < \frac{2}{\lambda_{\max}} \tag{28}$$

Because $\lambda_{\max}$ is the largest eigenvalue of the correlation sample $\boldsymbol{R_x}$

This can be difficult in reality.... then we use the trace instead

$$0 < \eta < \frac{2}{\text{trace}\left[\boldsymbol{R_x}\right]} \tag{29}$$

However, each diagonal element of $\boldsymbol{R_x}$ is equal the mean-squared value of the corresponding of the sensor input

We can re-state the previous condition as

$$0 < \eta < \frac{2}{\text{sum of the mean-square values of the sensor input}} \tag{30}$$

We get the following

The LMS is convergent in the mean square provided that η satisfies

$$0 < \eta < \frac{2}{\lambda_{\max}} \tag{28}$$

Because $\lambda_{\max}$ is the largest eigenvalue of the correlation sample $\boldsymbol{R}_{\boldsymbol{x}}$

This can be difficult in reality.... then we use the trace instead

$$0 < \eta < \frac{2}{\text{trace}\left[\boldsymbol{R}_{\boldsymbol{x}}\right]} \tag{29}$$

However, each diagonal element of $\boldsymbol{R}_{\boldsymbol{x}}$ is equal the mean-squared value of the corresponding of the sensor input

We can re-state the previous condition as

$$0 < \eta < \frac{2}{\text{sum of the mean-square values of the sensor input}} \tag{30}$$

Virtues and Limitations of the LMS Algorithm

Virtues

- An important virtue of the LMS algorithm is its simplicity.
- The model is independent and robust to the error (small disturbances = small estimation error).

Virtues and Limitations of the LMS Algorithm

Virtues

- An important virtue of the LMS algorithm is its simplicity.
- The model is independent and robust to the error (small disturbances = small estimation error).

Not only that, the LMS algorithm is optimal in accordance with the minimax criterion

If you do not know what you are up against, plan for the worst and optimize.

- The LMS algorithms requires about 10 times the dimensionality of the input space for convergence.

Virtues and Limitations of the LMS Algorithm

Virtues

- An important virtue of the LMS algorithm is its simplicity.
- The model is independent and robust to the error (small disturbances = small estimation error).

Not only that, the LMS algorithm is optimal in accordance with the minimax criterion

If you do not know what you are up against, plan for the worst and optimize.

Primary Limitation

- The slow rate of convergence and sensitivity to variations in the eigenstructure of the input.
- The LMS algorithms requires about 10 times the dimensionality of the input space for convergence.

Virtues and Limitations of the LMS Algorithm

Virtues

- An important virtue of the LMS algorithm is its simplicity.
- The model is independent and robust to the error (small disturbances = small estimation error).

Not only that, the LMS algorithm is optimal in accordance with the minimax criterion

If you do not know what you are up against, plan for the worst and optimize.

Primary Limitation

- The slow rate of convergence and sensitivity to variations in the eigenstructure of the input.
- The LMS algorithms requires about 10 times the dimensionality of the input space for convergence.

Virtues and Limitations of the LMS Algorithm

Virtues

- An important virtue of the LMS algorithm is its simplicity.
- The model is independent and robust to the error (small disturbances = small estimation error).

Not only that, the LMS algorithm is optimal in accordance with the minimax criterion

If you do not know what you are up against, plan for the worst and optimize.

Primary Limitation

- The slow rate of convergence and sensitivity to variations in the eigenstructure of the input.
- The LMS algorithms requires about 10 times the dimensionality of the input space for convergence.

More of this in...

Simon Haykin

Simon Haykin - Adaptive Filter Theory (3rd Edition)

Exercises

We have from NN by Haykin

3.1, 3.2, 3.3, 3.4, 3.5, 3.7 and 3.8

Outline

Cinvestav

Objective

Goal

Correctly classify a series of samples (External applied stimuli) $x_1, x_2, x_3, ..., x_m$ into one of two classes, C_1 and C_2.

Output of each input

1. Class C_1 output y $+1$.
2. Class C_2 output y -1.

Objective

Goal

Correctly classify a series of samples (External applied stimuli) $x_1, x_2, x_3, ..., x_m$ into one of two classes, C_1 and C_2.

Output of each input

1. Class C_1 output y $+1$.
2. Class C_2 output y -1.

History

Frank Rosenblatt

The perceptron algorithm was invented in 1957 at the Cornell Aeronautical Laboratory by Frank Rosenblatt.

Something Notable

Frank Rosenblatt was a Psychologist!!! Working at a militar R&D!!!

Frank Rosenblatt

He helped to develop the Mark I Perceptron - a new machine based in the connectivity of neural networks!!!

Some problems with it

- The most important is the impossibility to use the perceptron with a single neuron to solve the XOR problem

History

Frank Rosenblatt

The perceptron algorithm was invented in 1957 at the Cornell Aeronautical Laboratory by Frank Rosenblatt.

Something Notable

Frank Rosenblatt was a Psychologist!!! Working at a militar R&D!!!

Frank Rosenblatt

He helped to develop the Mark I Perceptron - a new machine based in the connectivity of neural networks!!!

Some problems with it

- The most important is the impossibility to use the perceptron with a single neuron to solve the XOR problem

History

Frank Rosenblatt

The perceptron algorithm was invented in 1957 at the Cornell Aeronautical Laboratory by Frank Rosenblatt.

Something Notable

Frank Rosenblatt was a Psychologist!!! Working at a militar R&D!!!

Frank Rosenblatt

He helped to develop the Mark I Perceptron - a new machine based in the connectivity of neural networks!!!

Some problems with it

- The most important is the impossibility to use the perceptron with a single neuron to solve the XOR problem

History

Frank Rosenblatt

The perceptron algorithm was invented in 1957 at the Cornell Aeronautical Laboratory by Frank Rosenblatt.

Something Notable

Frank Rosenblatt was a Psychologist!!! Working at a militar R&D!!!

Frank Rosenblatt

He helped to develop the Mark I Perceptron - a new machine based in the connectivity of neural networks!!!

Some problems with it

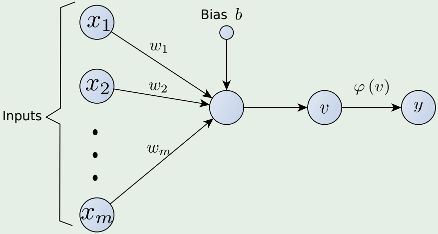
- The most important is the impossibility to use the perceptron with a single neuron to solve the XOR problem

Outline

Cinvestav

Perceptron: Local Field of a Neuron

Signal-Flow

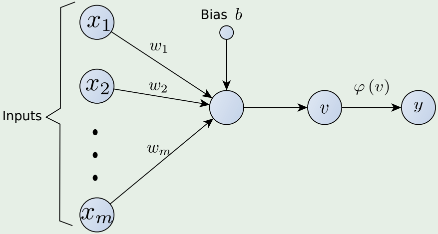


Induced local field of a neuron

$$v = \sum_{i=1}^{m} w_i x_i + b \tag{31}$$

Perceptron: Local Field of a Neuron

Signal-Flow



Induced local field of a neuron

$$v = \sum_{i=1}^{m} w_i x_i + b \tag{31}$$

Outline

Cinvestav

Perceptron: One Neuron Structure

Based in the previous induced local field

In the simplest form of the perceptron there are two decision regions separated by an **hyperplane:**

$$\sum_{i=1}^{m} w_i x_i + b = 0 \quad (32)$$

Example with two signals

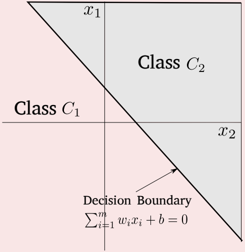
Perceptron: One Neuron Structure

Based in the previous induced local field

In the simplest form of the perceptron there are two decision regions separated by an **hyperplane:**

$$\sum_{i=1}^{m} w_i x_i + b = 0 \quad (32)$$

Example with two signals



Outline

Cinvestav

Deriving the Algorithm

First, you put signals together

$$x(n) = [1, x_1(n), x_2(n), ..., x_m(n)]^T \tag{33}$$

Weights

$$v(n) = \sum_{i=0}^{m} w_i(n) x_i(n) = w^T(n) x(n) \tag{34}$$

Note IMPORTANT - Perceptron works only if C_1 and C_2 are linearly separable

Deriving the Algorithm

First, you put signals together

$$x(n) = [1, x_1(n), x_2(n), ..., x_m(n)]^T \quad (33)$$

Weights

$$v(n) = \sum_{i=0}^{m} w_i(n) x_i(n) = \boldsymbol{w}^T(n) \boldsymbol{x}(n) \quad (34)$$

Note IMPORTANT - Perceptron works only if C_1 and C_2 are linearly separable

Deriving the Algorithm

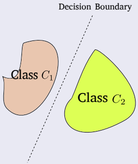
First, you put signals together

$$x(n) = [1, x_1(n), x_2(n), ..., x_m(n)]^T \quad (33)$$

Weights

$$v(n) = \sum_{i=0}^{m} w_i(n) x_i(n) = \boldsymbol{w}^T(n) \boldsymbol{x}(n) \quad (34)$$

Note IMPORTANT - Perceptron works only if C_1 and C_2 are linearly separable



Rule for Linear Separable Classes

There must exist a vector $\boldsymbol{w}$

1. $\boldsymbol{w}^T\boldsymbol{x} > 0$ for every input vector $\boldsymbol{x}$ belonging to class C_1.
2. $\boldsymbol{w}^T\boldsymbol{x} \leq 0$ for every input vector $\boldsymbol{x}$ belonging to class C_2.

Rule for Linear Separable Classes

There must exist a vector $\boldsymbol{w}$

1. $\boldsymbol{w}^T \boldsymbol{x} > 0$ for every input vector $\boldsymbol{x}$ belonging to class C_1.
2. $\boldsymbol{w}^T \boldsymbol{x} \leq 0$ for every input vector $\boldsymbol{x}$ belonging to class C_2.

What is the derivative of $\frac{dv(n)}{dw}$?

$$\frac{dv(n)}{dw} = \boldsymbol{x}(n) \quad (35)$$

Rule for Linear Separable Classes

There must exist a vector $\boldsymbol{w}$

1. $\boldsymbol{w}^T\boldsymbol{x} > 0$ for every input vector $\boldsymbol{x}$ belonging to class C_1.
2. $\boldsymbol{w}^T\boldsymbol{x} \leq 0$ for every input vector $\boldsymbol{x}$ belonging to class C_2.

What is the derivative of $\frac{dv(n)}{d\boldsymbol{w}}$?

$$\frac{dv(n)}{d\boldsymbol{w}} = \boldsymbol{x}(n) \tag{35}$$

Finally

No correction is necessary

1. $\boldsymbol{w}(n+1) = \boldsymbol{w}(n)$ if $\boldsymbol{w}^T\boldsymbol{x}(n) > 0$ and $\boldsymbol{x}(n)$ belongs to class C_1.
2. $\boldsymbol{w}(n+1) = \boldsymbol{w}(n)$ if and $\boldsymbol{w}^T\boldsymbol{x}(n) \leq 0$ and $\boldsymbol{x}(n) > 0$ belongs to class C_2.

Finally

No correction is necessary

1. $\boldsymbol{w}(n+1) = \boldsymbol{w}(n)$ if $\boldsymbol{w}^T\boldsymbol{x}(n) > 0$ and $\boldsymbol{x}(n)$ belongs to class C_1.
2. $\boldsymbol{w}(n+1) = \boldsymbol{w}(n)$ if and $\boldsymbol{w}^T\boldsymbol{x}(n) \leq 0$ and $\boldsymbol{x}(n) > 0$ belongs to class C_2.

Correction is necessary

1. $\boldsymbol{w}(n+1) = \boldsymbol{w}(n) - \eta(n)\,\boldsymbol{x}(n)$ if $\boldsymbol{w}^T(n)\,\boldsymbol{x}(n) > 0$ and $\boldsymbol{x}(n)$ belongs to class C_2.
2. $\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \eta(n)\,\boldsymbol{x}(n)$ if and $\boldsymbol{w}^T(n)\,\boldsymbol{x}(n) \leq 0$ and $\boldsymbol{x}(n)$ belongs to class C_1.

Where $\eta(n)$ is a learning parameter adjusting the learning rate.

Cinvestav

Finally

No correction is necessary

1. $\boldsymbol{w}(n+1) = \boldsymbol{w}(n)$ if $\boldsymbol{w}^T\boldsymbol{x}(n) > 0$ and $\boldsymbol{x}(n)$ belongs to class C_1.
2. $\boldsymbol{w}(n+1) = \boldsymbol{w}(n)$ if and $\boldsymbol{w}^T\boldsymbol{x}(n) \leq 0$ and $\boldsymbol{x}(n) > 0$ belongs to class C_2.

Correction is necessary

1. $\boldsymbol{w}(n+1) = \boldsymbol{w}(n) - \eta(n)\boldsymbol{x}(n)$ if $\boldsymbol{w}^T(n)\boldsymbol{x}(n) > 0$ and $\boldsymbol{x}(n)$ belongs to class C_2.
2. $\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \eta(n)\boldsymbol{x}(n)$ if and $\boldsymbol{w}^T(n)\boldsymbol{x}(n) \leq 0$ and $\boldsymbol{x}(n)$ belongs to class C_1.

Where $\eta(n)$ is a learning parameter adjusting the learning rate.

Cinvestav

Finally

No correction is necessary

1. $\boldsymbol{w}(n+1) = \boldsymbol{w}(n)$ if $\boldsymbol{w}^T\boldsymbol{x}(n) > 0$ and $\boldsymbol{x}(n)$ belongs to class C_1.
2. $\boldsymbol{w}(n+1) = \boldsymbol{w}(n)$ if and $\boldsymbol{w}^T\boldsymbol{x}(n) \leq 0$ and $\boldsymbol{x}(n) > 0$ belongs to class C_2.

Correction is necessary

1. $\boldsymbol{w}(n+1) = \boldsymbol{w}(n) - \eta(n)\,\boldsymbol{x}(n)$ if $\boldsymbol{w}^T(n)\,\boldsymbol{x}(n) > 0$ and $\boldsymbol{x}(n)$ belongs to class C_2.
2. $\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \eta(n)\,\boldsymbol{x}(n)$ if and $\boldsymbol{w}^T(n)\,\boldsymbol{x}(n) \leq 0$ and $\boldsymbol{x}(n)$ belongs to class C_1.

Where $\eta(n)$ is a learning parameter adjusting the learning rate.

Finally

No correction is necessary

1. $\boldsymbol{w}(n+1) = \boldsymbol{w}(n)$ if $\boldsymbol{w}^T\boldsymbol{x}(n) > 0$ and $\boldsymbol{x}(n)$ belongs to class C_1.
2. $\boldsymbol{w}(n+1) = \boldsymbol{w}(n)$ if and $\boldsymbol{w}^T\boldsymbol{x}(n) \leq 0$ and $\boldsymbol{x}(n) > 0$ belongs to class C_2.

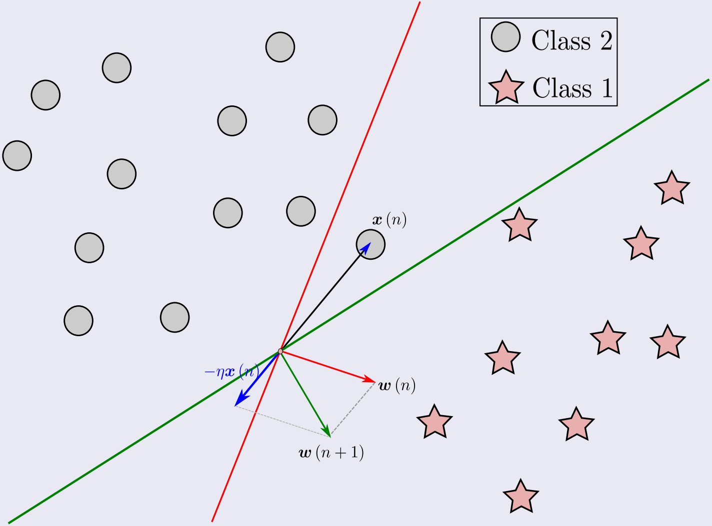
Correction is necessary

1. $\boldsymbol{w}(n+1) = \boldsymbol{w}(n) - \eta(n)\,\boldsymbol{x}(n)$ if $\boldsymbol{w}^T(n)\,\boldsymbol{x}(n) > 0$ and $\boldsymbol{x}(n)$ belongs to class C_2.
2. $\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \eta(n)\,\boldsymbol{x}(n)$ if and $\boldsymbol{w}^T(n)\,\boldsymbol{x}(n) \leq 0$ and $\boldsymbol{x}(n)$ belongs to class C_1.

Where $\eta(n)$ is a learning parameter adjusting the learning rate.

A little bit on the Geometry

For Example, $\boldsymbol{w}(n+1) = \boldsymbol{w}(n) - \eta\,(n)\,\boldsymbol{x}\,(n)$



Outline

Cinvestav

Under Linear Separability - Convergence happens!!!

If we assume

Linear Separabilty for the classes C_1 and C_2.

Rosenblatt - 1962

- Let the subsets of training vectors C_1 and C_2 be linearly separable.

Under Linear Separability - Convergence happens!!!

If we assume

Linear Separabilty for the classes C_1 and C_2.

Rosenblatt - 1962

- Let the subsets of training vectors C_1 and C_2 be linearly separable.
- Let the inputs presented to the perceptron originate from these two subsets.
- The perceptron converges after some n_0 iterations, in the sense that is a solution vector for

$$w(n_0) = w(n_0+1) = w(n_0+2) = \dots \quad (36)$$

is a solution vector for $n_0 \leq n_{max}$

Under Linear Separability - Convergence happens!!!

If we assume

Linear Separabilty for the classes C_1 and C_2.

Rosenblatt - 1962

- Let the subsets of training vectors C_1 and C_2 be linearly separable.
- Let the inputs presented to the perceptron originate from these two subsets.
- The perceptron converges after some n_0 iterations, in the sense that is a solution vector for

$$w(n_0) = w(n_0 + 1) = w(n_0 + 2) = \ldots \tag{36}$$

is a solution vector for $n_0 \leq n_{max}$

Cinvestav

Under Linear Separability - Convergence happens!!!

If we assume

Linear Separabilty for the classes C_1 and C_2.

Rosenblatt - 1962

- Let the subsets of training vectors C_1 and C_2 be linearly separable.
- Let the inputs presented to the perceptron originate from these two subsets.
- The perceptron converges after some n_0 iterations, in the sense that is a solution vector for

$$\boldsymbol{w}(n_0) = \boldsymbol{w}(n_0 + 1) = \boldsymbol{w}(n_0 + 2) = \dots \tag{36}$$

is a solution vector for $n_0 \leq n_{max}$

Outline

Cinvestav

Proof I - First a Lower Bound for $\|\boldsymbol{w}(n+1)\|^2$

Initialization

$$\boldsymbol{w}(0) = 0 \tag{37}$$

Proof I - First a Lower Bound for $\|\boldsymbol{w}(n+1)\|^2$

Initialization

$$\boldsymbol{w}(0) = 0 \tag{37}$$

Now assume for time $n = 1, 2, 3, ...$

$$\boldsymbol{w}^T(n)\,\boldsymbol{x}(n) < 0 \tag{38}$$

with $x(n)$ belongs to class C_1.

PERCEPTRON INCORRECTLY CLASSIFY THE VECTORS $x(1), x(2), ...$

Proof I - First a Lower Bound for $\left\|\boldsymbol{w}\left(n+1\right)\right\|^2$

Initialization

$$\boldsymbol{w}\left(0\right)=0 \tag{37}$$

Now assume for time $n=1,2,3,...$

$$\boldsymbol{w}^T\left(n\right)\boldsymbol{x}\left(n\right)<0 \tag{38}$$

with $\boldsymbol{x}(n)$ belongs to class C_1.

PERCEPTRON INCORRECTLY CLASSIFY THE VECTORS $\boldsymbol{x}\left(1\right),\boldsymbol{x}\left(2\right),...$

Apply the correction formula

$$\boldsymbol{w}\left(n+1\right)=\boldsymbol{w}\left(n\right)+\boldsymbol{x}\left(n\right) \tag{39}$$

Proof I - First a Lower Bound for $\|\boldsymbol{w}(n+1)\|^2$

Initialization

$$\boldsymbol{w}(0) = 0 \tag{37}$$

Now assume for time $n = 1, 2, 3, \ldots$

$$\boldsymbol{w}^T(n)\,\boldsymbol{x}(n) < 0 \tag{38}$$

with $\boldsymbol{x}(n)$ belongs to class C_1.

PERCEPTRON INCORRECTLY CLASSIFY THE VECTORS $\boldsymbol{x}(1), \boldsymbol{x}(2), \ldots$

Apply the correction formula

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \boldsymbol{x}(n) \tag{39}$$

Proof I - First a Lower Bound for $\|\boldsymbol{w}(n+1)\|^2$

Initialization

$$\boldsymbol{w}(0) = 0 \quad (37)$$

Now assume for time $n = 1, 2, 3, ...$

$$\boldsymbol{w}^T(n)\,\boldsymbol{x}(n) < 0 \quad (38)$$

with $\boldsymbol{x}(n)$ belongs to class C_1.

PERCEPTRON INCORRECTLY CLASSIFY THE VECTORS $\boldsymbol{x}(1), \boldsymbol{x}(2), ...$

Apply the correction formula

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \boldsymbol{x}(n) \quad (39)$$

Proof II

Apply the correction iteratively

$$\boldsymbol{w}(n+1) = \boldsymbol{x}(1) + \boldsymbol{x}(2) + \ldots + \boldsymbol{x}(n) \tag{40}$$

We know that there is a solution w_0(Linear Separability)

$$\alpha = \min_{\boldsymbol{x}(n) \in C_1} \boldsymbol{w}_0^T \boldsymbol{x}(n) \tag{41}$$

Then, we have

$$\boldsymbol{w}_0^T \boldsymbol{w}(n+1) = \boldsymbol{w}_0^T \boldsymbol{x}(1) + \boldsymbol{w}_0^T \boldsymbol{x}(2) + \ldots + \boldsymbol{w}_0^T \boldsymbol{x}(n) \tag{42}$$

Cinvestav

Proof II

Apply the correction iteratively

$$\boldsymbol{w}(n+1) = \boldsymbol{x}(1) + \boldsymbol{x}(2) + \ldots + \boldsymbol{x}(n) \tag{40}$$

We know that there is a solution $\boldsymbol{w}_0$(Linear Separability)

$$\alpha = \min_{\boldsymbol{x}(n) \in C_1} \boldsymbol{w}_0^T \boldsymbol{x}(n) \tag{41}$$

Then, we have

$$\boldsymbol{w}_0^T \boldsymbol{w}(n+1) = \boldsymbol{w}_0^T \boldsymbol{x}(1) + \boldsymbol{w}_0^T \boldsymbol{x}(2) + \ldots + \boldsymbol{w}_0^T \boldsymbol{x}(n) \tag{42}$$

Cinvestav

Proof II

Apply the correction iteratively

$$\boldsymbol{w}(n+1) = \boldsymbol{x}(1) + \boldsymbol{x}(2) + \ldots + \boldsymbol{x}(n) \tag{40}$$

We know that there is a solution $\boldsymbol{w}_0$(Linear Separability)

$$\alpha = \min_{\boldsymbol{x}(n) \in C_1} \boldsymbol{w}_0^T \boldsymbol{x}(n) \tag{41}$$

Then, we have

$$\boldsymbol{w}_{\boldsymbol{0}}^{\boldsymbol{T}} \boldsymbol{w}(n+1) = \boldsymbol{w}_{\boldsymbol{0}}^{\boldsymbol{T}} \boldsymbol{x}(1) + \boldsymbol{w}_0^T \boldsymbol{x}(2) + \ldots + \boldsymbol{w}_{\boldsymbol{0}}^{\boldsymbol{T}} \boldsymbol{x}(n) \tag{42}$$

Cinvestav

Proof III

Apply the correction iteratively

$$\boldsymbol{w}(n+1) = \boldsymbol{x}(1) + \boldsymbol{x}(2) + ... + \boldsymbol{x}(n) \tag{43}$$

We know that there is a solution w_0(Linear Separability)

$$\alpha = \min_{\boldsymbol{x}(n) \in C_1} w_0^T \boldsymbol{x}(n) \tag{44}$$

Then, we have

$$w_0^T \boldsymbol{w}(n+1) = w_0^T \boldsymbol{x}(1) + w_0^T \boldsymbol{x}(2) + ... + w_0^T \boldsymbol{x}(n) \tag{45}$$

Cinvestav

Proof III

Apply the correction iteratively

$$\boldsymbol{w}(n+1) = \boldsymbol{x}(1) + \boldsymbol{x}(2) + \ldots + \boldsymbol{x}(n) \tag{43}$$

We know that there is a solution $\boldsymbol{w}_0$(Linear Separability)

$$\alpha = \min_{\boldsymbol{x}(n) \in C_1} \boldsymbol{w}_0^T \boldsymbol{x}(n) \tag{44}$$

Then, we have

$$\boldsymbol{w}_0^T \boldsymbol{w}(n+1) = \boldsymbol{w}_0^T \boldsymbol{x}(1) + \boldsymbol{w}_0^T \boldsymbol{x}(2) + \ldots + \boldsymbol{w}_0^T \boldsymbol{x}(n) \tag{45}$$

Proof III

Apply the correction iteratively

$$\boldsymbol{w}(n+1) = \boldsymbol{x}(1) + \boldsymbol{x}(2) + ... + \boldsymbol{x}(n) \tag{43}$$

We know that there is a solution $\boldsymbol{w}_0$(Linear Separability)

$$\alpha = \min_{\boldsymbol{x}(n) \in C_1} \boldsymbol{w}_0^T \boldsymbol{x}(n) \tag{44}$$

Then, we have

$$\boldsymbol{w}_{\boldsymbol{0}}^{\boldsymbol{T}} \boldsymbol{w}(n+1) = \boldsymbol{w}_{\boldsymbol{0}}^{\boldsymbol{T}} \boldsymbol{x}(1) + \boldsymbol{w}_0^T \boldsymbol{x}(2) + ... + \boldsymbol{w}_{\boldsymbol{0}}^{\boldsymbol{T}} \boldsymbol{x}(n) \tag{45}$$

Proof IV

Thus we use the α

$$\boldsymbol{w}_{\boldsymbol{0}}^{\boldsymbol{T}}\boldsymbol{w}\left(n+1\right)\geq n\alpha \quad (46)$$

Thus using the Cauchy-Schwartz Inequality

$$\left\|w_0^T\right\|^2\left\|w\left(n+1\right)\right\|^2\geq\left[w_0^T w\left(n+1\right)\right]^2 \quad (47)$$

$\|\cdot\|$ is the Euclidean distance.

Thus

$$\left\|w_0^T\right\|^2\left\|w\left(n+1\right)\right\|^2 \geq n^2\alpha^2$$

$$\left\|w\left(n+1\right)\right\|^2 \geq \frac{n^2\alpha^2}{\left\|w_0^T\right\|^2}$$

Proof IV

Thus we use the α

$$\boldsymbol{w}_{\boldsymbol{0}}^{\boldsymbol{T}}\boldsymbol{w}(n+1) \geq n\alpha \tag{46}$$

Thus using the Cauchy-Schwartz Inequality

$$\left\|\boldsymbol{w}_0^T\right\|^2 \|\boldsymbol{w}(n+1)\|^2 \geq \left[\boldsymbol{w}_{\boldsymbol{0}}^{\boldsymbol{T}}\boldsymbol{w}(n+1)\right]^2 \tag{47}$$

$\|\cdot\|$ is the Euclidean distance.

Thus

$$\begin{aligned}\left\|\boldsymbol{w}_0^T\right\|^2 \|\boldsymbol{w}(n+1)\|^2 &\geq n^2\alpha^2\\ \|\boldsymbol{w}(n+1)\|^2 &\geq \frac{n^2\alpha^2}{\left\|\boldsymbol{w}_0^T\right\|^2}\end{aligned}$$

Proof IV

Thus we use the α

$$\boldsymbol{w}_{\mathbf{0}}^{\boldsymbol{T}}\boldsymbol{w}\left(n+1\right) \geq n\alpha \tag{46}$$

Thus using the Cauchy-Schwartz Inequality

$$\left\|\boldsymbol{w}_0^T\right\|^2 \left\|\boldsymbol{w}\left(n+1\right)\right\|^2 \geq \left[\boldsymbol{w}_{\mathbf{0}}^{\boldsymbol{T}}\boldsymbol{w}\left(n+1\right)\right]^2 \tag{47}$$

$\|\cdot\|$ is the Euclidean distance.

Thus

$$\begin{aligned}\left\|\boldsymbol{w}_0^T\right\|^2 \left\|\boldsymbol{w}\left(n+1\right)\right\|^2 &\geq n^2\alpha^2 \\ \left\|\boldsymbol{w}\left(n+1\right)\right\|^2 &\geq \frac{n^2\alpha^2}{\left\|\boldsymbol{w}_0^T\right\|^2}\end{aligned}$$

Proof V - Now a Upper Bound for $\|\boldsymbol{w}(n+1)\|^2$

Now rewrite equation 39

$$\boldsymbol{w}(k+1) = \boldsymbol{w}(k) + \boldsymbol{x}(k) \tag{48}$$

for $k = 1, 2, ..., n$ and $\boldsymbol{x}(k) \in C_1$

Squaring the Euclidean norm of both sides

$$\|\boldsymbol{w}(k+1)\|^2 = \|\boldsymbol{w}(k)\|^2 + \|\boldsymbol{x}(k)\|^2 + 2\boldsymbol{w}^T(k)\boldsymbol{x}(k) \tag{49}$$

Now taking that $\boldsymbol{w}^T(k)\boldsymbol{x}(k) < 0$

$$\begin{aligned} \|\boldsymbol{w}(k+1)\|^2 &\leq \|\boldsymbol{w}(k)\|^2 + \|\boldsymbol{x}(k)\|^2 \\ \|\boldsymbol{w}(k+1)\|^2 - \|\boldsymbol{w}(k)\|^2 &\leq \|\boldsymbol{x}(k)\|^2 \end{aligned}$$

Cinvestav

Proof V - Now a Upper Bound for $\|\boldsymbol{w}(n+1)\|^2$

Now rewrite equation 39

$$\boldsymbol{w}(k+1) = \boldsymbol{w}(k) + \boldsymbol{x}(k) \tag{48}$$

for $k = 1, 2, ..., n$ and $\boldsymbol{x}(k) \in C_1$

Squaring the Euclidean norm of both sides

$$\|\boldsymbol{w}(k+1)\|^2 = \|\boldsymbol{w}(k)\|^2 + \|\boldsymbol{x}(k)\|^2 + 2\boldsymbol{w}^T(k)\boldsymbol{x}(k) \tag{49}$$

Now taking that $\boldsymbol{w}^T(k)\boldsymbol{x}(k) < 0$

$$\begin{aligned} \|\boldsymbol{w}(k+1)\|^2 &\leq \|\boldsymbol{w}(k)\|^2 + \|\boldsymbol{x}(k)\|^2 \\ \|\boldsymbol{w}(k+1)\|^2 - \|\boldsymbol{w}(k)\|^2 &\leq \|\boldsymbol{x}(k)\|^2 \end{aligned}$$

Proof V - Now a Upper Bound for $\left\|\boldsymbol{w}\left(n+1\right)\right\|^2$

Now rewrite equation 39

$$\boldsymbol{w}\left(k+1\right)=\boldsymbol{w}\left(k\right)+\boldsymbol{x}\left(k\right) \tag{48}$$

for $k=1,2,...,n$ and $\boldsymbol{x}\left(k\right)\in C_1$

Squaring the Euclidean norm of both sides

$$\left\|\boldsymbol{w}\left(k+1\right)\right\|^2=\left\|\boldsymbol{w}\left(k\right)\right\|^2+\left\|\boldsymbol{x}\left(k\right)\right\|^2+2\boldsymbol{w}^T\left(k\right)\boldsymbol{x}\left(k\right) \tag{49}$$

Now taking that $\boldsymbol{w}^T\left(k\right)\boldsymbol{x}\left(k\right)<0$

$$\begin{aligned}\left\|\boldsymbol{w}\left(k+1\right)\right\|^2 &\leq \left\|\boldsymbol{w}\left(k\right)\right\|^2+\left\|\boldsymbol{x}\left(k\right)\right\|^2\\ \left\|\boldsymbol{w}\left(k+1\right)\right\|^2-\left\|\boldsymbol{w}\left(k\right)\right\|^2 &\leq \left\|\boldsymbol{x}\left(k\right)\right\|^2\end{aligned}$$

Proof VI

Use the telescopic sum

$$\sum_{k=0}^{n}\left[\|\boldsymbol{w}(k+1)\|^2-\|\boldsymbol{w}(k)\|^2\right]\leq\sum_{k=0}^{n}\|\boldsymbol{x}(k)\|^2 \tag{50}$$

Assume

$$\begin{aligned}\boldsymbol{w}(0) &= 0\\ \boldsymbol{x}(0) &= 0\end{aligned}$$

Thus

$$\|\boldsymbol{w}(n+1)\|^2 \leq \sum_{k=1}^{n}\|\boldsymbol{x}(k)\|^2$$

Cinvestav

Proof VI

Use the telescopic sum

$$\sum_{k=0}^{n}\left[\left\|\boldsymbol{w}\left(k+1\right)\right\|^{2}-\left\|\boldsymbol{w}\left(k\right)\right\|^{2}\right]\leq\sum_{k=0}^{n}\left\|\boldsymbol{x}\left(k\right)\right\|^{2} \quad (50)$$

Assume

$$\begin{aligned}\boldsymbol{w}\left(0\right) &= \mathbf{0}\\ \boldsymbol{x}\left(0\right) &= \mathbf{0}\end{aligned}$$

Thus

$$\left\|\boldsymbol{w}\left(n+1\right)\right\|^{2} \leq \sum_{k=1}^{n}\left\|\boldsymbol{x}\left(k\right)\right\|^{2}$$

Proof VI

Use the telescopic sum

$$\sum_{k=0}^{n}\left[\|\boldsymbol{w}(k+1)\|^2 - \|\boldsymbol{w}(k)\|^2\right] \leq \sum_{k=0}^{n}\|\boldsymbol{x}(k)\|^2 \tag{50}$$

Assume

$$\begin{aligned}\boldsymbol{w}(0) &= \mathbf{0}\\ \boldsymbol{x}(0) &= \mathbf{0}\end{aligned}$$

Thus

$$\|\boldsymbol{w}(n+1)\|^2 \leq \sum_{k=1}^{n} \|x(k)\|^2$$

Cinvestav

Proof VII

Then, we can define a positive number

$$\beta = \max_{\boldsymbol{x}(k) \in C_1} \|\boldsymbol{x}(k)\|^2 \tag{51}$$

Thus

$$\|w(k+1)\|^2 \leq \sum_{k=1}^{n} \|x(k)\|^2 \leq n\beta$$

Thus, we satisfies the equations only when exists a n_{max} (Using Our Sandwich)

$$\frac{n_{max}^2 \alpha^2}{\|w_0\|^2} = n_{max}\beta \tag{52}$$

Proof VII

Then, we can define a positive number

$$\beta = \max_{\boldsymbol{x}(k)\in C_1} \left\|\boldsymbol{x}(k)\right\|^2 \tag{51}$$

Thus

$$\left\|\boldsymbol{w}(k+1)\right\|^2 \leq \sum_{k=1}^{n} \left\|x(k)\right\|^2 \leq n\beta$$

Thus, we satisfies the equations only when exists a n_{max} (Using Our Sandwich)

$$\frac{n_{max}^2 \alpha^2}{\left\|w_0\right\|^2} = n_{max}\beta \tag{52}$$

Cinvestav

Proof VII

Then, we can define a positive number

$$\beta = \max_{\boldsymbol{x}(k)\in C_1} \left\|\boldsymbol{x}(k)\right\|^2 \quad (51)$$

Thus

$$\left\|\boldsymbol{w}(k+1)\right\|^2 \leq \sum_{k=1}^{n} \left\|x(k)\right\|^2 \leq n\beta$$

Thus, we satisfies the equations only when exists a n_{max} (Using Our Sandwich)

$$\frac{n_{max}^2\alpha^2}{\left\|\boldsymbol{w}_0\right\|^2} = n_{max}\beta \quad (52)$$

Cinvestav

Proof VIII

Solving

$$n_{max} = \frac{\beta \left\| \boldsymbol{w}_0 \right\|^2}{\alpha^2} \quad (53)$$

Thus

For $\eta(n) = 1$ for all n, $w(0) = 0$ and a solution vector w_0:

- The rule for adaptying the synaptic weights of the perceptron must terminate after at most n_{max} steps.

In addition

Because w_0 the solution is not unique.

Proof VIII

Solving

$$n_{max} = \frac{\beta \left\| \boldsymbol{w}_0 \right\|^2}{\alpha^2} \tag{53}$$

Thus

For $\eta(n) = 1$ for all n, $\boldsymbol{w}(0) = \mathbf{0}$ and a solution vector $\boldsymbol{w}_0$:

- The rule for adaptying the synaptic weights of the perceptron must terminate after at most n_{max} steps.

In addition

Because w_0 the solution is not unique.

Cinvestav

Proof VIII

Solving

$$n_{max} = \frac{\beta \left\| \boldsymbol{w}_0 \right\|^2}{\alpha^2} \quad (53)$$

Thus

For $\eta(n) = 1$ for all n, $\boldsymbol{w}(0) = \mathbf{0}$ and a solution vector $\boldsymbol{w}_0$:

- The rule for adaptying the synaptic weights of the perceptron must terminate after at most n_{max} steps.

In addition

Because w_0 the solution is not unique.

Outline

Cinvestav

Algorithm Using Error-Correcting

Now, if we use the $\frac{1}{2}e_k(n)^2$

We can actually simplify the rules and the final algorithm!!!

Thus, we have the following Delta Value

$$\Delta w(n) = \eta((d_j - y_j(n)))x(n) \tag{54}$$

Cinvestav

Algorithm Using Error-Correcting

Now, if we use the $\frac{1}{2}e_k(n)^2$

We can actually simplify the rules and the final algorithm!!!

Thus, we have the following Delta Value

$$\Delta \boldsymbol{w}(n) = \eta\left(\left(d_j - y_j(n)\right)\right)\boldsymbol{x}(n) \tag{54}$$

Outline

Cinvestav

We could use the previous Rule

In order to generate an algorithm

However, you need classes that are linearly separable!!!

Therefore, we can use a more generals Gradient Descent Rule

To obtain an algorithm to the best separation hyperplane!!!

Cinvestav

We could use the previous Rule

In order to generate an algorithm

However, you need classes that are linearly separable!!!

Therefore, we can use a more generals Gradient Descent Rule

To obtain an algorithm to the best separation hyperplane!!!

Gradient Descent Algorithm

Algorithm - Off-line/Batch Learning

1. Set $n = 0$.
2. Set $d_j = \begin{cases} +1 & \text{if } x_j(n) \in \text{Class 1} \\ -1 & \text{if } x_j(n) \in \text{Class 2} \end{cases}$ for all $j = 1, 2, ..., m$.
3. Initialize the weights, $w^T = (w_1(n), w_2(n), ..., w_n(n))$.
   - Weights may be initialized to 0 or to a small random value.
4. Initialize Dummy outputs so you can enter loop $y^t = \langle y_1(n), y_2(n), ..., y_m(n)\rangle$
5. Initialize Stopping error $\epsilon > 0$.
6. Initialize learning error η.
7. While $\frac{1}{m}\sum_{j=1}^{m} \|d_j - y_j(n)\| > \epsilon$
   - For each sample (x_j, d_j) for $j = 1, ..., m$:
     - Calculate output $y_j = \varphi\left(\boldsymbol{w}^T(n) \cdot \boldsymbol{x}_j\right)$
     - Update weights $w_i(n+1) = w_i(n) + \eta(d_j - y_j(n))\, x_{ij}$.
   - $n = n + 1$

Gradient Descent Algorithm

Algorithm - Off-line/Batch Learning

1. Set $n = 0$.
2. Set $d_j = \begin{cases} +1 & \text{if } \boldsymbol{x}_j(n) \in \text{Class 1} \\ -1 & \text{if } \boldsymbol{x}_j(n) \in \text{Class 2} \end{cases}$ for all $j = 1, 2, ..., m$.
3. Initialize the weights, $\boldsymbol{w}^T = (w_1(n), w_2(n), ..., w_n(n))$.
   - Weights may be initialized to 0 or to a small random value.
4. Initialize Dummy outputs so you can enter loop $\boldsymbol{y}^t = \langle y_1(n)., y_2(n), ..., y_m(n)\rangle$
5. Initialize Stopping error $\epsilon > 0$.
6. Initialize learning error η.
7. While $\frac{1}{m}\sum_{j=1}^{m} \|d_j - y_j(n)\| > \epsilon$
   - For each sample (x_j, d_j) for $j = 1, ..., m$:
     - ★ Calculate output $y_j = \varphi\left(\boldsymbol{w}^T(n) \cdot \boldsymbol{x}_j\right)$
     - ★ Update weights $w_i(n+1) = w_i(n) + \eta(d_j - y_j(n))x_{ij}$.
   - $n = n + 1$

Nevertheless

We have the following problem

$$\epsilon > 0 \quad (55)$$

Thus...

Convergence to the best linear separation is a tweaking business!!!

Cinvestav

Nevertheless

We have the following problem

$$\epsilon > 0 \tag{55}$$

Thus...

Convergence to the best linear separation is a tweaking business!!!

Cinvestav

Outline

Cinvestav

However, if we limit our features!!!

The Winnow Algorithm!!!

It converges even with no-linear separability.

Feature Vector

A Boolean-valued features $X = \{0, 1\}^d$

Weight Vector w

1. $w^t = (w_1, w_2, \ldots, w_p)$ for all $w_i \in \mathbb{R}$
2. For all i, $w_i \geq 0$.

However, if we limit our features!!!

The Winnow Algorithm!!!

It converges even with no-linear separability.

Feature Vector

A Boolean-valued features $X = \{0, 1\}^d$

Weight Vector w

1. $w^t = (w_1, w_2, ..., w_p)$ for all $w_i \in \mathbb{R}$
2. For all i, $w_i \geq 0$.

However, if we limit our features!!!

The Winnow Algorithm!!!

It converges even with no-linear separability.

Feature Vector

A Boolean-valued features $X = \{0, 1\}^d$

Weight Vector $\boldsymbol{w}$

1. $\boldsymbol{w}^t = (w_1, w_2, ..., w_p)$ for all $w_i \in \mathbb{R}$
2. For all i, $w_i \geq 0$.

Classification Scheme

We use a specific θ

1. $\boldsymbol{w}^T\boldsymbol{x} \geq \theta \Rightarrow$ positive classification Class 1 if $\boldsymbol{x}$ $\in$Class 1
2. $\boldsymbol{w}^T\boldsymbol{x} < \theta \Rightarrow$ negative classification Class 2 if $\boldsymbol{x}$ $\in$Class 2

Rule

We use two possible Rules for training!!! With a learning rate of $\alpha > 1$.

Rule 1

- When misclassifying a positive training example $\boldsymbol{x}$ $\in$Class 1 i.e. $\boldsymbol{w}^T\boldsymbol{x} < \theta$

$$\forall x_i = 1 : w_i \leftarrow \alpha w_i \quad (56)$$

Classification Scheme

We use a specific θ

1. $\boldsymbol{w}^T\boldsymbol{x} \geq \theta \Rightarrow$ positive classification Class 1 if $\boldsymbol{x} \in$Class 1
2. $\boldsymbol{w}^T\boldsymbol{x} < \theta \Rightarrow$ negative classification Class 2 if $\boldsymbol{x} \in$Class 2

Rule

We use two possible Rules for training!!! With a learning rate of $\alpha > 1$.

Rule 1

- When misclassifying a positive training example $x \in$Class 1 i.e. $\boldsymbol{w}^T\boldsymbol{x} < \theta$

$$\forall x_i = 1 : w_i \leftarrow \alpha w_i \quad (56)$$

Classification Scheme

We use a specific θ

1. $\boldsymbol{w}^T\boldsymbol{x} \geq \theta \Rightarrow$ positive classification Class 1 if $\boldsymbol{x} \in$Class 1
2. $\boldsymbol{w}^T\boldsymbol{x} < \theta \Rightarrow$ negative classification Class 2 if $\boldsymbol{x} \in$Class 2

Rule

We use two possible Rules for training!!! With a learning rate of $\alpha > 1$.

Rule 1

- When misclassifying a positive training example $\boldsymbol{x} \in$Class 1 i.e. $\boldsymbol{w}^T\boldsymbol{x} < \theta$

$$\forall x_i = 1 : w_i \leftarrow \alpha w_i \tag{56}$$

Classification Scheme

Rule 2

- When misclassifying a negative training example $\boldsymbol{x} \in$Class 2 i.e. $\boldsymbol{w}^T\boldsymbol{x} \geq \theta$

$$\forall x_i = 1 : w_i \leftarrow \frac{w_i}{\alpha} \tag{57}$$

Rule 3

- If samples are correctly classified do nothing!!!

Classification Scheme

Rule 2

- When misclassifying a negative training example $\boldsymbol{x}$ $\in$Class 2 i.e. $\boldsymbol{w}^T\boldsymbol{x} \geq \theta$

$$\forall x_i = 1 : w_i \leftarrow \frac{w_i}{\alpha} \quad (57)$$

Rule 3

- If samples are correctly classified do nothing!!!

Cinvestav

Properties of Winnow

Property

- If there are many irrelevant variables Winnow is better than the Perceptron.

Drawback

- Sensitive to the learning rate α.

Properties of Winnow

Property

- If there are many irrelevant variables Winnow is better than the Perceptron.

Drawback

- Sensitive to the learning rate α.

The Pocket Algorithm

A variant of the Perceptron Algorithm

It was suggested by Geman et al. in

- "Perceptron based learning algorithms," IEEE Transactions on Neural Networks,Vol. 1(2), pp. 179–191, 1990.
- It converges to an optimal solution even if the linear separability is not fulfilled.

Cinvestav

The Pocket Algorithm

A variant of the Perceptron Algorithm

It was suggested by Geman et al. in

- "Perceptron based learning algorithms," IEEE Transactions on Neural Networks,Vol. 1(2), pp. 179–191, 1990.
- It converges to an optimal solution even if the linear separability is not fulfilled.

Cinvestav

The Pocket Algorithm

A variant of the Perceptron Algorithm

It was suggested by Geman et al. in

- "Perceptron based learning algorithms," IEEE Transactions on Neural Networks,Vol. 1(2), pp. 179–191, 1990.
- It converges to an optimal solution even if the linear separability is not fulfilled.

Finally

It consists of the following steps

1. Initialize the weight vector $\boldsymbol{w}(0)$ in a random way.
2. Define a storage pocket vector $\boldsymbol{w}_s$ and a history counter h_s to zero for the same pocket vector.
3. At the i^{th} iteration step compute the update $\boldsymbol{w}(n+1)$ using the Perceptron rule.
4. Use the update weight to find the number h of samples correctly classified.
5. If at any moment $h > h_s$ replace $\boldsymbol{w}_s$ with $\boldsymbol{w}(n+1)$ and h_s with h
6. Keep iterating to 3 until convergence.
7. Return $\boldsymbol{w}_s$.

Finally

It consists of the following steps

1. Initialize the weight vector $\boldsymbol{w}(0)$ in a random way.
2. Define a storage pocket vector $\boldsymbol{w}_s$ and a history counter h_s to zero for the same pocket vector.
3. At the i^{th} iteration step compute the update $\boldsymbol{w}(n+1)$ using the Perceptron rule.
4. Use the update weight to find the number h of samples correctly classified.
5. If at any moment $h > h_s$ replace $\boldsymbol{w}_s$ with $\boldsymbol{w}(n+1)$ and h_s with h
6. Keep iterating to 3 until convergence.
7. Return $\boldsymbol{w}_s$.

Finally

It consists of the following steps

1. Initialize the weight vector $\boldsymbol{w}(0)$ in a random way.
2. Define a storage pocket vector $\boldsymbol{w}_s$ and a history counter h_s to zero for the same pocket vector.
3. At the i^{th} iteration step compute the update $\boldsymbol{w}(n+1)$ using the Perceptron rule.
4. Use the update weight to find the number h of samples correctly classified.
5. If at any moment $h > h_s$ replace $\boldsymbol{w}_s$ with $\boldsymbol{w}(n+1)$ and h_s with h
6. Keep iterating to 3 until convergence.
7. Return $\boldsymbol{w}_s$.

Finally

It consists of the following steps

1. Initialize the weight vector $\boldsymbol{w}(0)$ in a random way.
2. Define a storage pocket vector $\boldsymbol{w}_s$ and a history counter h_s to zero for the same pocket vector.
3. At the i^{th} iteration step compute the update $\boldsymbol{w}(n+1)$ using the Perceptron rule.
4. Use the update weight to find the number h of samples correctly classified.
5. If at any moment $h > h_s$ replace $\boldsymbol{w}_s$ with $\boldsymbol{w}(n+1)$ and h_s with h
6. Keep iterating to 3 until convergence.
7. Return $\boldsymbol{w}_s$.

Cinvestav

Finally

It consists of the following steps

1. Initialize the weight vector $\boldsymbol{w}(0)$ in a random way.
2. Define a storage pocket vector $\boldsymbol{w}_s$ and a history counter h_s to zero for the same pocket vector.
3. At the i^{th} iteration step compute the update $\boldsymbol{w}(n+1)$ using the Perceptron rule.
4. Use the update weight to find the number h of samples correctly classified.
5. If at any moment $h > h_s$ replace $\boldsymbol{w}_s$ with $\boldsymbol{w}(n+1)$ and h_s with h
6. Keep iterating to 3 until convergence.
7. Return $\boldsymbol{w}_s$.

Cinvestav

Finally

It consists of the following steps

1. Initialize the weight vector $\boldsymbol{w}(0)$ in a random way.
2. Define a storage pocket vector $\boldsymbol{w}_s$ and a history counter h_s to zero for the same pocket vector.
3. At the i^{th} iteration step compute the update $\boldsymbol{w}(n+1)$ using the Perceptron rule.
4. Use the update weight to find the number h of samples correctly classified.
5. If at any moment $h > h_s$ replace $\boldsymbol{w}_s$ with $\boldsymbol{w}(n+1)$ and h_s with h
6. Keep iterating to 3 until convergence.
7. Return $\boldsymbol{w}_s$.

Cinvestav

Finally

It consists of the following steps

1. Initialize the weight vector $\boldsymbol{w}(0)$ in a random way.
2. Define a storage pocket vector $\boldsymbol{w}_s$ and a history counter h_s to zero for the same pocket vector.
3. At the i^{th} iteration step compute the update $\boldsymbol{w}(n+1)$ using the Perceptron rule.
4. Use the update weight to find the number h of samples correctly classified.
5. If at any moment $h > h_s$ replace $\boldsymbol{w}_s$ with $\boldsymbol{w}(n+1)$ and h_s with h
6. Keep iterating to 3 until convergence.
7. Return $\boldsymbol{w}_s$.

Cinvestav