

# Introduction to Neural Networks and Deep Learning

## Introduction Single-Layer Perceptron

Andres Mendez-Vazquez

September 23, 2019

# Outline

## 1 Introduction

- History

## 2 Adapting Filtering Problem

- Definition
- Description of the Behavior of the System

## 3 Unconstrained Optimization

- Introduction
- Method of Steepest Descent
- Newton's Method
- Gauss-Newton Method

## 4 Linear Least-Squares Filter

- Introduction
- Least-Mean-Square (LMS) Algorithm

## 5 Perceptron

- Objective
- Perceptron: Local Field of a Neuron
- Perceptron: One Neuron Structure
- Deriving the Algorithm
- Under Linear Separability - Convergence happens!!!
- Proof
- Algorithm Using Error-Correcting
- Final Perceptron Algorithm (One Version)
- The Winnow Algorithm
- Parallel Implementations



# Outline

## 1 Introduction

- History

## 2 Adapting Filtering Problem

- Definition
- Description of the Behavior of the System

## 3 Unconstrained Optimization

- Introduction
- Method of Steepest Descent
- Newton's Method
- Gauss-Newton Method

## 4 Linear Least-Squares Filter

- Introduction
- Least-Mean-Square (LMS) Algorithm

## 5 Perceptron

- Objective
- Perceptron: Local Field of a Neuron
- Perceptron: One Neuron Structure
- Deriving the Algorithm
- Under Linear Separability - Convergence happens!!!
- Proof
- Algorithm Using Error-Correcting
- Final Perceptron Algorithm (One Version)
- The Winnow Algorithm
- Parallel Implementations



## At the beginning of Neural Networks (1943 - 1958)

- McCulloch and Pitts (1943) [1] for introducing the idea of neural networks as computing machines.
- Hebb (1949) [2] for postulating the first rule for self-organized learning.
- Rosenblatt (1958) [3] for proposing the perceptron as the first model for learning with a teacher (i.e., supervised learning).



## At the beginning of Neural Networks (1943 - 1958)

- McCulloch and Pitts (1943) [1] for introducing the idea of neural networks as computing machines.
- Hebb (1949) [2] for postulating the first rule for self-organized learning.
- Rosenblatt (1958) [3] 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 classification of patterns said to be linearly separable (i.e., patterns that lie on opposite sides of a hyperplane).



## At the beginning of Neural Networks (1943 - 1958)

- McCulloch and Pitts (1943) [1] for introducing the idea of neural networks as computing machines.
- Hebb (1949) [2] for postulating the first rule for self-organized learning.
- Rosenblatt (1958) [3] 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 classification of patterns said to be linearly separable (i.e., patterns that lie on opposite sides of a hyperplane).



## At the beginning of Neural Networks (1943 - 1958)

- McCulloch and Pitts (1943) [1] for introducing the idea of neural networks as computing machines.
- Hebb (1949) [2] for postulating the first rule for self-organized learning.
- Rosenblatt (1958) [3] 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 classification 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.

### Background

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

## 1 Introduction

- History

## 2 Adapting Filtering Problem

- **Definition**
- Description of the Behavior of the System

## 3 Unconstrained Optimization

- Introduction
- Method of Steepest Descent
- Newton's Method
- Gauss-Newton Method

## 4 Linear Least-Squares Filter

- Introduction
- Least-Mean-Square (LMS) Algorithm

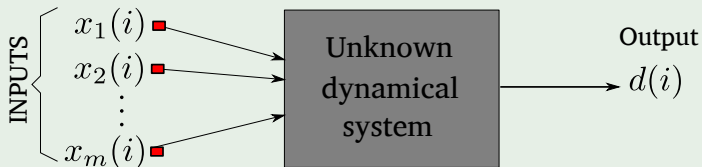
## 5 Perceptron

- Objective
- Perceptron: Local Field of a Neuron
- Perceptron: One Neuron Structure
- Deriving the Algorithm
- Under Linear Separability - Convergence happens!!!
- Proof
- Algorithm Using Error-Correcting
- Final Perceptron Algorithm (One Version)
- The Winnow Algorithm
- Parallel Implementations



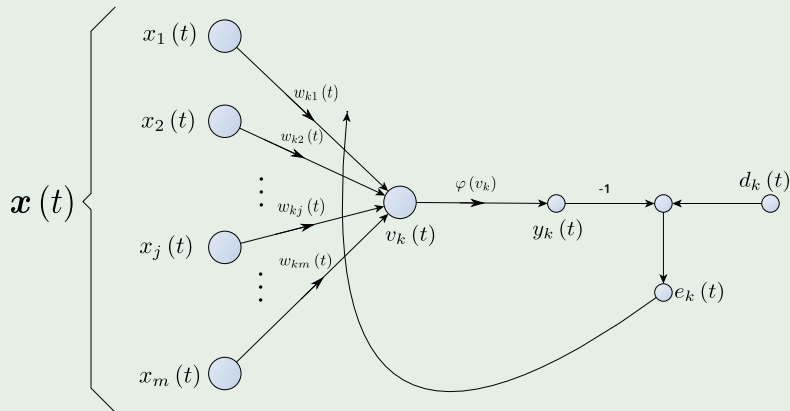
# Adapting Filtering Problem

Consider a dynamical system



# Signal-Flow Graph of Adaptive Model

We have the following equivalence



# Outline

## 1 Introduction

- History

## 2 Adapting Filtering Problem

- Definition
- **Description of the Behavior of the System**

## 3 Unconstrained Optimization

- Introduction
- Method of Steepest Descent
- Newton's Method
- Gauss-Newton Method

## 4 Linear Least-Squares Filter

- Introduction
- Least-Mean-Square (LMS) Algorithm

## 5 Perceptron

- Objective
- Perceptron: Local Field of a Neuron
- Perceptron: One Neuron Structure
- Deriving the Algorithm
- Under Linear Separability - Convergence happens!!!
- Proof
- Algorithm Using Error-Correcting
- Final Perceptron Algorithm (One Version)
- The Winnow Algorithm
- Parallel Implementations



# Description of the Behavior of the System

We have the data set

$$\mathcal{T} = \{(\mathbf{x}(i), d(i)) \mid i = 1, 2, \dots, n, \dots\} \quad (1)$$

Where

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



# Description of the Behavior of the System

We have the data set

$$\mathcal{T} = \{(\mathbf{x}(i), d(i)) \mid i = 1, 2, \dots, n, \dots\} \quad (1)$$

Where

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

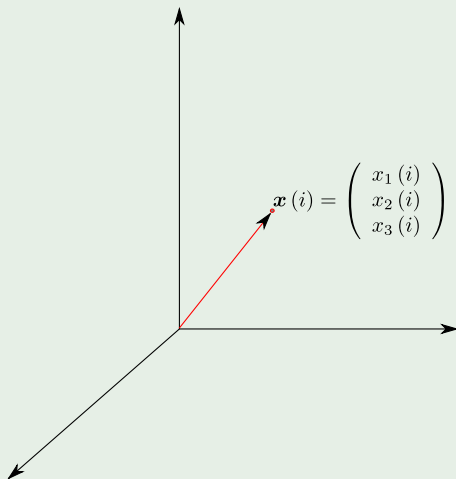




## The Stimulus $\mathbf{x}(i)$

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

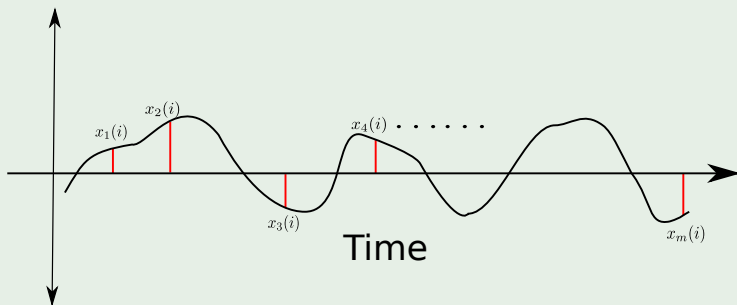
The  $m$  elements of  $\mathbf{x}(i)$  originate at different points in space (spatial)



# The Stimulus $x(i)$

The stimulus  $x(i)$  can arise from

The  $m$  elements of  $x(i)$  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?

Yes, 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?

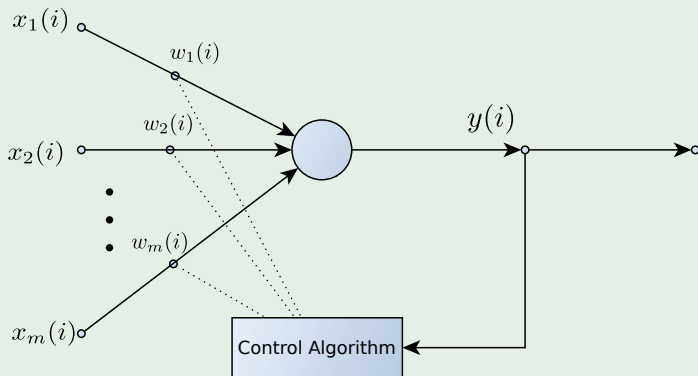
## It is more

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 algorithm 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 algorithm 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 algorithm 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

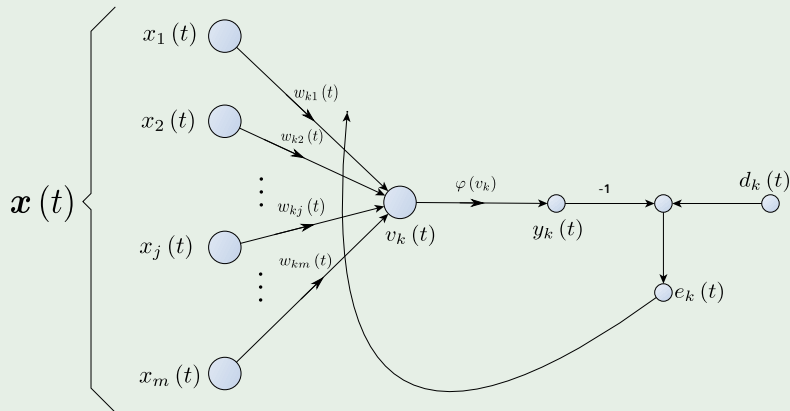
Computation of adjustments to synaptic weights are completed inside a time interval that is one sampling period long.





# Signal-Flow Graph of Adaptive Model

We have the following equivalence



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

### Filtering processes

- 1 An output, denoted by  $y(i)$ , that is produced in response to the  $m$  elements of the stimulus vector  $\mathbf{x}(i)$ .
- 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  $e(i)$

### Summary

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 continuous processes

### Filtering processes

- 1 An output, denoted by  $y(i)$ , that is produced in response to the  $m$  elements of the stimulus vector  $\mathbf{x}(i)$ .
- 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  $e(i)$

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 continuous processes

### Filtering processes

- 1 An output, denoted by  $y(i)$ , that is produced in response to the  $m$  elements of the stimulus vector  $\mathbf{x}(i)$ .
- 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  $v(i)$

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

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

$$y(i) = x^T(i) w(i) \quad (4)$$

Error

$$e(i) = d(i) - y(i) \quad (5)$$



Thus

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

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

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

$$y(i) = \mathbf{x}^T(i) \mathbf{w}(i) \quad (4)$$

Error

$$e(i) = d(i) - y(i) \quad (5)$$



# Thus

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

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

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

$$y(i) = \mathbf{x}^T(i) \mathbf{w}(i) \quad (4)$$

## Error

$$e(i) = d(i) - y(i) \quad (5)$$



# Outline

## 1 Introduction

- History

## 2 Adapting Filtering Problem

- Definition
- Description of the Behavior of the System

## 3 Unconstrained Optimization

### ● Introduction

- Method of Steepest Descent
- Newton's Method
- Gauss-Newton Method

## 4 Linear Least-Squares Filter

- Introduction
- Least-Mean-Square (LMS) Algorithm

## 5 Perceptron

- Objective
- Perceptron: Local Field of a Neuron
- Perceptron: One Neuron Structure
- Deriving the Algorithm
- Under Linear Separability - Convergence happens!!!
- Proof
- Algorithm Using Error-Correcting
- Final Perceptron Algorithm (One Version)
- The Winnow Algorithm
- Parallel Implementations





## Consider

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

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

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

We want to

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

For this

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



## Consider

A continuous differentiable function  $J(\mathbf{w})$

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

$$J(\mathbf{w}^*) \leq J(\mathbf{w}), \forall \mathbf{w} \quad (6)$$

We want to

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

For this

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



## Consider

A continuous differentiable function  $J(\mathbf{w})$

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

$$J(\mathbf{w}^*) \leq J(\mathbf{w}), \forall \mathbf{w} \quad (6)$$

We want to

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

For this

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



## Where

$\nabla$  is the gradient operator

$$\nabla = \left[ \frac{\partial}{\partial w_1}, \frac{\partial}{\partial w_2}, \dots, \frac{\partial}{\partial w_m} \right]^T \quad (8)$$

This

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



## Where

$\nabla$  is the gradient operator

$$\nabla = \left[ \frac{\partial}{\partial w_1}, \frac{\partial}{\partial w_2}, \dots, \frac{\partial}{\partial w_m} \right]^T \quad (8)$$

Thus

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



Thus

Starting with an initial guess denoted by  $w(0)$ ,

Then, generate a sequence of weight vectors  $w(1), w(2), \dots$

Such that you can reduce  $J(w)$  at each iteration

$$J(w(n+1)) < J(w(n)) \quad (10)$$

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



Thus

Starting with an initial guess denoted by  $w(0)$ ,

Then, generate a sequence of weight vectors  $w(1), w(2), \dots$

Such that you can reduce  $J(w)$  at each iteration

$$J(w(n+1)) < J(w(n)) \quad (10)$$

**Where:**  $w(n)$  is the old value and  $w(n+1)$  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

## 1 Introduction

- History

## 2 Adapting Filtering Problem

- Definition
- Description of the Behavior of the System

## 3 Unconstrained Optimization

- Introduction
- **Method of Steepest Descent**
- Newton's Method
- Gauss-Newton Method

## 4 Linear Least-Squares Filter

- Introduction
- Least-Mean-Square (LMS) Algorithm

## 5 Perceptron

- Objective
- Perceptron: Local Field of a Neuron
- Perceptron: One Neuron Structure
- Deriving the Algorithm
- Under Linear Separability - Convergence happens!!!
- Proof
- Algorithm Using Error-Correcting
- Final Perceptron Algorithm (One Version)
- The Winnow Algorithm
- Parallel Implementations



## Steepest Descent [4]

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

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

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

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

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

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



## Steepest Descent [4]

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

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

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

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

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

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



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

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

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

Example



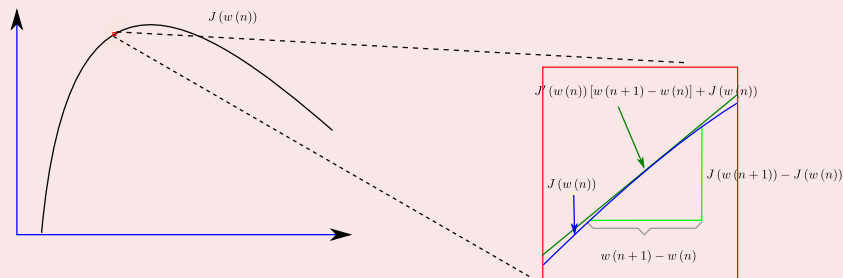
Cinvestav

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

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

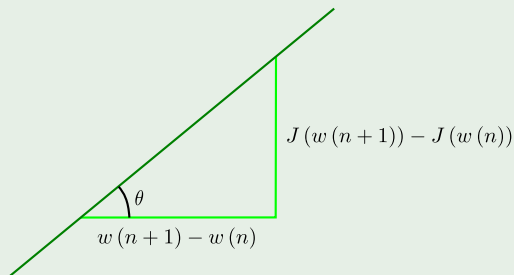
$$L(w(n)) = J'(w(n)) [w(n+1) - w(n)] + J(w(n)) \quad (12)$$

### Example



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

## Remember Something quite Classic



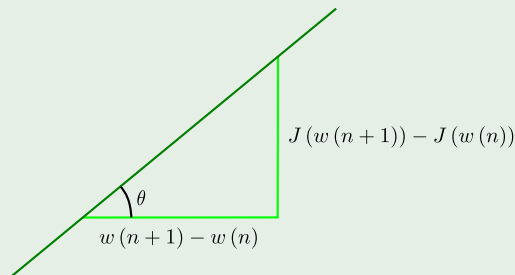
$$\tan \theta = \frac{J(w(n+1)) - J(w(n))}{w(n+1) - w(n)}$$

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

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

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

## Remember Something quite Classic



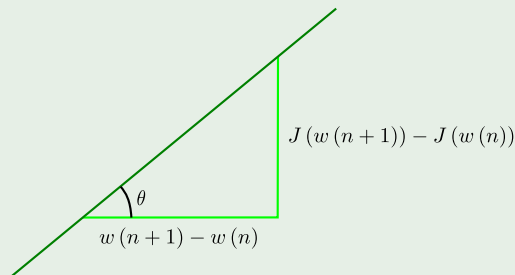
$$\tan \theta = \frac{J(w(n+1)) - J(w(n))}{w(n+1) - w(n)}$$

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

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

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

## Remember Something quite Classic



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

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

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



Thus, we have that

Using the First Taylor expansion

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



## Now, for Many Variables

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

$$H = \{ \mathbf{x} \mid \mathbf{a}^T \mathbf{x} = b \} \quad (14)$$

Given  $x \in H$  and  $x_0 \in H$

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

Thus, we have that

$$H = \{ \mathbf{x} \mid \mathbf{a}^T (\mathbf{x} - \mathbf{x}_0) = 0 \}$$



## Now, for Many Variables

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

$$H = \{ \mathbf{x} \mid \mathbf{a}^T \mathbf{x} = b \} \quad (14)$$

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

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

Thus, we have that

$$H = \{ \mathbf{x} \mid \mathbf{a}^T (\mathbf{x} - \mathbf{x}_0) = 0 \}$$



## Now, for Many Variables

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

$$H = \{ \mathbf{x} \mid \mathbf{a}^T \mathbf{x} = b \} \quad (14)$$

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

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

Thus, we have that

$$H = \{ \mathbf{x} \mid \mathbf{a}^T (\mathbf{x} - \mathbf{x}_0) = 0 \}$$



Thus, we have the following definition

### Definition (Differentiability)

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

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



Thus, we have the following definition

### Definition (Differentiability)

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

①  $\frac{\partial J(\mathbf{w}(n))}{\partial w_i}$  exist for all  $i = 1, \dots, n$ .

②  $J$  is locally linear at  $\mathbf{w}(n)$ .



Thus, we have the following definition

### Definition (Differentiability)

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

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



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

We know that we have the following operator

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

Thus, we have

$$\nabla J(\mathbf{w}(n)) = \left( \frac{\partial J(\mathbf{w}(n))}{\partial w_1}, \frac{\partial J(\mathbf{w}(n))}{\partial w_2}, \dots, \frac{\partial J(\mathbf{w}(n))}{\partial w_m} \right)$$





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

We know that we have the following operator

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

Thus, we have

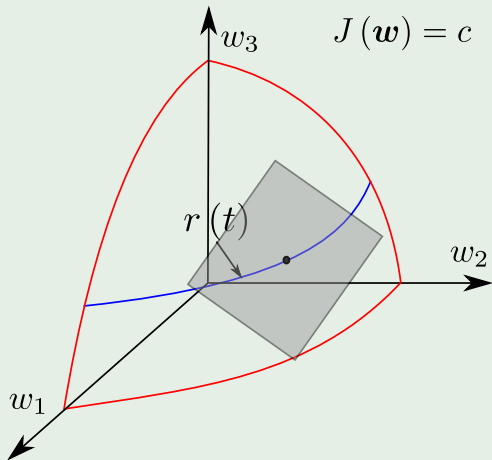
$$\begin{aligned} \nabla J(\mathbf{w}(n)) &= \left( \frac{\partial J(\mathbf{w}(n))}{\partial w_1}, \frac{\partial J(\mathbf{w}(n))}{\partial w_2}, \dots, \frac{\partial J(\mathbf{w}(n))}{\partial w_m} \right) \\ &= \sum_{i=1}^m \hat{w}_i \frac{\partial J(\mathbf{w}(n))}{\partial w_i} \end{aligned}$$

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



Now

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



# Level Set

## Definition

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

**Remark:** In a normal Calculus course we will use  $x$  and  $f$  instead of  $w$  and  $J$ .



## Where

Any curve has the following parametrization

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

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

We can write the parametrized version of it

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

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

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

## Where

Any curve has the following parametrization

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

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

We can write the parametrized version of it

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

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

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

## Where

Any curve has the following parametrization

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

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

We can write the parametrized version of it

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

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

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

# Note

## First

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

We have then that

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

Thus

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

## Note

### First

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

We have then that

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

This

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



## Note

### First

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

### We have then that

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

### Thus

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

Thus

Evaluating at  $t = n$

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

We have that

$$\nabla J(\mathbf{w}(n)) \cdot \mathbf{r}'(n) = 0 \quad (20)$$

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  $\mathbf{w}(n)$ .



Thus

Evaluating at  $t = n$

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

We have that

$$\nabla J(\mathbf{w}(n)) \cdot \mathbf{r}'(n) = 0 \quad (20)$$

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  $\mathbf{w}(n)$ .



Thus

Evaluating at  $t = n$

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

We have that

$$\nabla J(\mathbf{w}(n)) \cdot \mathbf{r}'(n) = 0 \quad (20)$$

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  $\mathbf{w}(n)$ .



Now the tangent plane to the surface can be described generally

Thus

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

This looks like

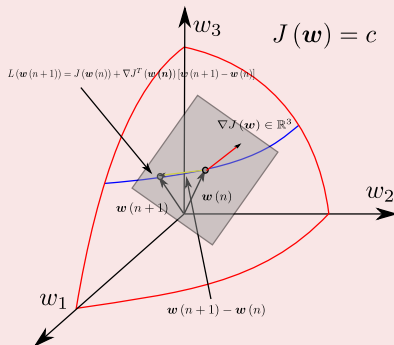


Now the tangent plane to the surface can be described generally

Thus

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

This looks like



# Proving the fact about the Steepest Descent

We want the following

$$J(\mathbf{w}(n+1)) < J(\mathbf{w}(n))$$

Using the first-order Taylor approximation

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

So, we ask the following

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



# Proving the fact about the Steepest Descent

We want the following

$$J(\mathbf{w}(n+1)) < J(\mathbf{w}(n))$$

Using the first-order Taylor approximation

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

So, we ask the following

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





# Proving the fact about the Steepest Descent

We want the following

$$J(\mathbf{w}(n+1)) < J(\mathbf{w}(n))$$

Using the first-order Taylor approximation

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

So, we ask the following

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



Then

We have that

$$J(\mathbf{w}(n+1)) - J(\mathbf{w}(n)) \approx -\eta \nabla J^T(\mathbf{w}(n)) \nabla J(\mathbf{w}(n)) = -\eta \|\nabla J(\mathbf{w}(n))\|^2$$

Thus

$$J(\mathbf{w}(n+1)) - J(\mathbf{w}(n)) < 0$$

Or

$$J(\mathbf{w}(n+1)) < J(\mathbf{w}(n))$$



Then

We have that

$$J(\mathbf{w}(n+1)) - J(\mathbf{w}(n)) \approx -\eta \nabla J^T(\mathbf{w}(n)) \nabla J(\mathbf{w}(n)) = -\eta \|\nabla J(\mathbf{w}(n))\|^2$$

Thus

$$J(\mathbf{w}(n+1)) - J(\mathbf{w}(n)) < 0$$

Or

$$J(\mathbf{w}(n+1)) < J(\mathbf{w}(n))$$



Then

We have that

$$J(\mathbf{w}(n+1)) - J(\mathbf{w}(n)) \approx -\eta \nabla J^T(\mathbf{w}(n)) \nabla J(\mathbf{w}(n)) = -\eta \|\nabla J(\mathbf{w}(n))\|^2$$

Thus

$$J(\mathbf{w}(n+1)) - J(\mathbf{w}(n)) < 0$$

Or

$$J(\mathbf{w}(n+1)) < J(\mathbf{w}(n))$$



# Outline

## 1 Introduction

- History

## 2 Adapting Filtering Problem

- Definition
- Description of the Behavior of the System

## 3 Unconstrained Optimization

- Introduction
- Method of Steepest Descent
- **Newton's Method**
- Gauss-Newton Method

## 4 Linear Least-Squares Filter

- Introduction
- Least-Mean-Square (LMS) Algorithm

## 5 Perceptron

- Objective
- Perceptron: Local Field of a Neuron
- Perceptron: One Neuron Structure
- Deriving the Algorithm
- Under Linear Separability - Convergence happens!!!
- Proof
- Algorithm Using Error-Correcting
- Final Perceptron Algorithm (One Version)
- The Winnow Algorithm
- Parallel Implementations



# Newton's Method

## Here

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

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

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

where given that  $\mathbf{w}(n)$  is a vector with dimension  $w$ ,

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

# Newton's Method

## Here

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

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

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

where given that  $\Delta \mathbf{w}(n)$  is a vector with dimension  $m$

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

# Newton's Method

## Here

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

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

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

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

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



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

Do you have any idea?

Look again

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

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

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

Thus

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



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

Do you have any idea?

Look again

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

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

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

This

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



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

Do you have any idea?

Look again

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

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

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

Thus

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



# The Final Method

Define the following

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

Then

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



# The Final Method

Define the following

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

Then

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



# Outline

## 1 Introduction

- History

## 2 Adapting Filtering Problem

- Definition
- Description of the Behavior of the System

## 3 Unconstrained Optimization

- Introduction
- Method of Steepest Descent
- Newton's Method
- **Gauss-Newton Method**

## 4 Linear Least-Squares Filter

- Introduction
- Least-Mean-Square (LMS) Algorithm

## 5 Perceptron

- Objective
- Perceptron: Local Field of a Neuron
- Perceptron: One Neuron Structure
- Deriving the Algorithm
- Under Linear Separability - Convergence happens!!!
- Proof
- Algorithm Using Error-Correcting
- Final Perceptron Algorithm (One Version)
- The Winnow Algorithm
- Parallel Implementations



We have then an error

## Something Notable

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

Thus using the first order Taylor expansion

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

In matrix form

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



We have then an error

### Something Notable

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

Thus using the first order Taylor expansion

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

in matrix form

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





We have then an error

### Something Notable

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

Thus using the first order Taylor expansion

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

In matrix form

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



## Where

The error vector is equal to

$$\mathbf{e}(n) = [e(1), e(2), \dots, e(n)]^T \quad (24)$$

Thus, we get the famous Jacobian once we derive:

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



## Where

The error vector is equal to

$$\mathbf{e}(n) = [e(1), e(2), \dots, e(n)]^T \quad (24)$$

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

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



# Where

We want the following

$$\mathbf{w}(n+1) = \underset{\mathbf{w}}{\operatorname{arg\,min}} \left\{ \frac{1}{2} \|\mathbf{e}_l(n, \mathbf{w})\|^2 \right\}$$

Idea:

What if we expand out the equation?



# Where

We want the following

$$\mathbf{w}(n+1) = \underset{\mathbf{w}}{\operatorname{arg\,min}} \left\{ \frac{1}{2} \|\mathbf{e}_l(n, \mathbf{w})\|^2 \right\}$$

## Ideas

What if we expand out the equation?



# Expanded Version

We get

$$\begin{aligned} \frac{1}{2} \|e_l(n, \mathbf{w})\|^2 &= \frac{1}{2} \|e(n)\|^2 + e^T(n) \nabla J(n) (\mathbf{w} - \mathbf{w}(n)) + \dots \\ &\quad \frac{1}{2} (\mathbf{w} - \mathbf{w}(n))^T \nabla^T J(n) \nabla J(n) (\mathbf{w} - \mathbf{w}(n)) \end{aligned}$$



Therefore

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

$$\nabla^T J(n) \mathbf{e}(n) + \nabla^T J(n) \nabla J(n) [\mathbf{w} - \mathbf{w}(n)] = 0$$

We get finally

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



Therefore

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

$$\nabla^T J(n) \mathbf{e}(n) + \nabla^T J(n) \nabla J(n) [\mathbf{w} - \mathbf{w}(n)] = 0$$

We get finally

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





## 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  $e(n)$ .

## However

The Gauss-Newton iteration to be computable, the matrix product  $\nabla^T J(n) \nabla J(n)$  must be nonsingular.



## 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  $e(n)$ .

## However

The Gauss-Newton iteration to be computable, the matrix product  $\nabla^T J(n) \nabla J(n)$  must be nonsingular.



# Outline

## 1 Introduction

- History

## 2 Adapting Filtering Problem

- Definition
- Description of the Behavior of the System

## 3 Unconstrained Optimization

- Introduction
- Method of Steepest Descent
- Newton's Method
- Gauss-Newton Method

## 4 Linear Least-Squares Filter

- **Introduction**
- Least-Mean-Square (LMS) Algorithm

## 5 Perceptron

- Objective
- Perceptron: Local Field of a Neuron
- Perceptron: One Neuron Structure
- Deriving the Algorithm
- Under Linear Separability - Convergence happens!!!
- Proof
- Algorithm Using Error-Correcting
- Final Perceptron Algorithm (One Version)
- The Winnow Algorithm
- Parallel Implementations



# Introduction

A linear least-squares filter has two distinctive characteristics [5]

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

Thus, expressing the error

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

short version - error is linear in the weight vector  $\mathbf{w}(n)$

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

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

# Introduction

A linear least-squares filter has two distinctive characteristics [5]

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

Thus, expressing the error

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

short version - error is linear in the weight vector  $\mathbf{w}(n)$

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

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

# Introduction

A linear least-squares filter has two distinctive characteristics [5]

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

Thus, expressing the error

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

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

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

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

Now, differentiate  $e(n)$  with respect to  $w(n)$

Thus

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

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

$$\nabla J(n) = -\mathbf{X}(n)$$

Let us to use the Gauss-Newton

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



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

Thus

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

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

$$\nabla J(n) = -\mathbf{X}(n)$$

Let us now use the Gauss-Newton

$$\mathbf{w}(n+1) = \mathbf{w}(n) - \left( \nabla^T J(n) \nabla J(n) \right)^{-1} \nabla^T J(n) e(n)$$





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

Thus

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

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

$$\nabla J(n) = -\mathbf{X}(n)$$

Let us to use the Gaussian-Newton

$$\mathbf{w}(n+1) = \mathbf{w}(n) - \left( \nabla^T J(n) \nabla J(n) \right)^{-1} \nabla^T J(n) e(n)$$



Thus

We have the following

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

We have then

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

Thus, we have

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

Thus

We have the following

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

We have then

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

Thus, we have

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

Thus

We have the following

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

We have then

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

Thus, we have

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

# Outline

## 1 Introduction

- History

## 2 Adapting Filtering Problem

- Definition
- Description of the Behavior of the System

## 3 Unconstrained Optimization

- Introduction
- Method of Steepest Descent
- Newton's Method
- Gauss-Newton Method

## 4 Linear Least-Squares Filter

- Introduction
- **Least-Mean-Square (LMS) Algorithm**

## 5 Perceptron

- Objective
- Perceptron: Local Field of a Neuron
- Perceptron: One Neuron Structure
- Deriving the Algorithm
- Under Linear Separability - Convergence happens!!!
- Proof
- Algorithm Using Error-Correcting
- Final Perceptron Algorithm (One Version)
- The Winnow Algorithm
- Parallel Implementations



## Again Our Error Cost function

We have

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

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

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

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

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

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

## Again Our Error Cost function

We have

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

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

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

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

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

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

## Again Our Error Cost function

We have

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

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

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

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

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

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



We have

## Something Notable

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

Then

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

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

$$\hat{\mathbf{w}}(n+1) = \hat{\mathbf{w}}(n) + \eta \mathbf{x}(n) e(n) \quad (27)$$



We have

### Something Notable

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

Then

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

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

$$\hat{\mathbf{w}}(n+1) = \hat{\mathbf{w}}(n) + \eta \mathbf{x}(n) e(n) \quad (27)$$



We have

Something Notable

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

Then

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

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

$$\hat{\mathbf{w}}(n+1) = \hat{\mathbf{w}}(n) + \eta \mathbf{x}(n) e(n) \quad (27)$$



## 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.



## 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

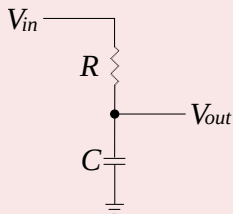


## Remarks

### 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



# Learning of the data

## Thus

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



# Learning of the data

## Thus

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

## Assigning a small value to $\eta$ ,

- The adaptive process progresses slowly.





# Learning of the data

## Thus

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

## Assigning a small value to $\eta$ ,

- The adaptive process progresses slowly.

## More of the past data is used by the LMS algorithm

- The more the LMS is a more accurate filter.



# 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  $\Rightarrow$  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.

# 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.

## Principal 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)

# Outline

## 1 Introduction

- History

## 2 Adapting Filtering Problem

- Definition
- Description of the Behavior of the System

## 3 Unconstrained Optimization

- Introduction
- Method of Steepest Descent
- Newton's Method
- Gauss-Newton Method

## 4 Linear Least-Squares Filter

- Introduction
- Least-Mean-Square (LMS) Algorithm

## 5 Perceptron

- Objective
- Perceptron: Local Field of a Neuron
- Perceptron: One Neuron Structure
- Deriving the Algorithm
- Under Linear Separability - Convergence happens!!!
- Proof
- Algorithm Using Error-Correcting
- Final Perceptron Algorithm (One Version)
- The Winnow Algorithm
- Parallel Implementations





# Objective

## Goal

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

## Output of each input

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



# Objective

## Goal

Correctly classify a series of samples (External applied stimuli)  $x_1, x_2, x_3, \dots, x_m$  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

## 1 Introduction

- History

## 2 Adapting Filtering Problem

- Definition
- Description of the Behavior of the System

## 3 Unconstrained Optimization

- Introduction
- Method of Steepest Descent
- Newton's Method
- Gauss-Newton Method

## 4 Linear Least-Squares Filter

- Introduction
- Least-Mean-Square (LMS) Algorithm

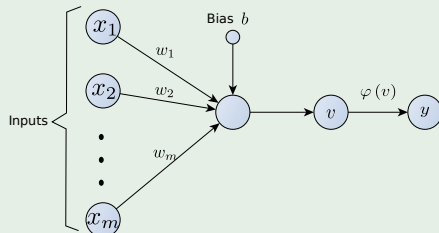
## 5 Perceptron

- Objective
- **Perceptron: Local Field of a Neuron**
- Perceptron: One Neuron Structure
- Deriving the Algorithm
- Under Linear Separability - Convergence happens!!!
- Proof
- Algorithm Using Error-Correcting
- Final Perceptron Algorithm (One Version)
- The Winnow Algorithm
- Parallel Implementations



# Perceptron: Local Field of a Neuron

## Signal-Flow



Induced local field of a neuron

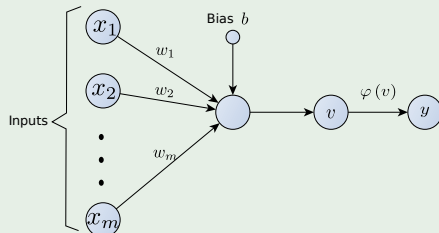
$$v = \sum_{i=1}^m w_i x_i + b \quad (28)$$





# Perceptron: Local Field of a Neuron

## Signal-Flow



## Induced local field of a neuron

$$v = \sum_{i=1}^m w_i x_i + b \quad (28)$$



# Outline

## 1 Introduction

- History

## 2 Adapting Filtering Problem

- Definition
- Description of the Behavior of the System

## 3 Unconstrained Optimization

- Introduction
- Method of Steepest Descent
- Newton's Method
- Gauss-Newton Method

## 4 Linear Least-Squares Filter

- Introduction
- Least-Mean-Square (LMS) Algorithm

## 5 Perceptron

- Objective
- Perceptron: Local Field of a Neuron
- **Perceptron: One Neuron Structure**
- Deriving the Algorithm
- Under Linear Separability - Convergence happens!!!
- Proof
- Algorithm Using Error-Correcting
- Final Perceptron Algorithm (One Version)
- The Winnow Algorithm
- Parallel Implementations



# 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**:

$$\sum_{i=1}^m w_i x_i + b = 0 \quad (29)$$

Example with two signals



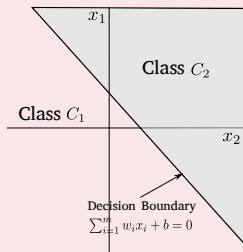
# Perceptron: One Neuron Structure

Based in the previous induced local field

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

$$\sum_{i=1}^m w_i x_i + b = 0 \quad (29)$$

Example with two signals



# Outline

## 1 Introduction

- History

## 2 Adapting Filtering Problem

- Definition
- Description of the Behavior of the System

## 3 Unconstrained Optimization

- Introduction
- Method of Steepest Descent
- Newton's Method
- Gauss-Newton Method

## 4 Linear Least-Squares Filter

- Introduction
- Least-Mean-Square (LMS) Algorithm

## 5 Perceptron

- Objective
- Perceptron: Local Field of a Neuron
- Perceptron: One Neuron Structure
- **Deriving the Algorithm**
- Under Linear Separability - Convergence happens!!!
- Proof
- Algorithm Using Error-Correcting
- Final Perceptron Algorithm (One Version)
- The Winnow Algorithm
- Parallel Implementations



# Deriving the Algorithm

First, you put signals together

$$x(n) = [1, x_1(n), x_2(n), \dots, x_m(n)]^T \quad (30)$$

Weights

$$v(n) = \sum_{i=0}^m w_i(n) x_i(n) = w^T(n) x(n) \quad (31)$$

NOTE: IMPORTANT – Perceptron only is adaptive, and only for linearly separable



# Deriving the Algorithm

First, you put signals together

$$\mathbf{x}(n) = [1, x_1(n), x_2(n), \dots, x_m(n)]^T \quad (30)$$

Weights

$$v(n) = \sum_{i=0}^m w_i(n) x_i(n) = \mathbf{w}^T(n) \mathbf{x}(n) \quad (31)$$

NOT IMPORTANT – Perception and cognition are not necessarily sequential



# Deriving the Algorithm

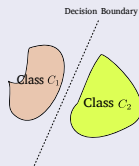
First, you put signals together

$$\mathbf{x}(n) = [1, x_1(n), x_2(n), \dots, x_m(n)]^T \quad (30)$$

Weights

$$v(n) = \sum_{i=0}^m w_i(n) x_i(n) = \mathbf{w}^T(n) \mathbf{x}(n) \quad (31)$$

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





# Rule for Linear Separable Classes

There must exist a vector  $w$

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

What is the derivative of  $v(n)$ ?

$$\frac{dv(n)}{dw} = x(n) \quad (32)$$



# Rule for Linear Separable Classes

There must exist a vector  $w$

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

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

$$\frac{dv(n)}{dw} = x(n) \quad (32)$$



# Finally

## No correction is necessary

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

## Correction is necessary

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

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



# Finally

## No correction is necessary

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

## Correction is necessary

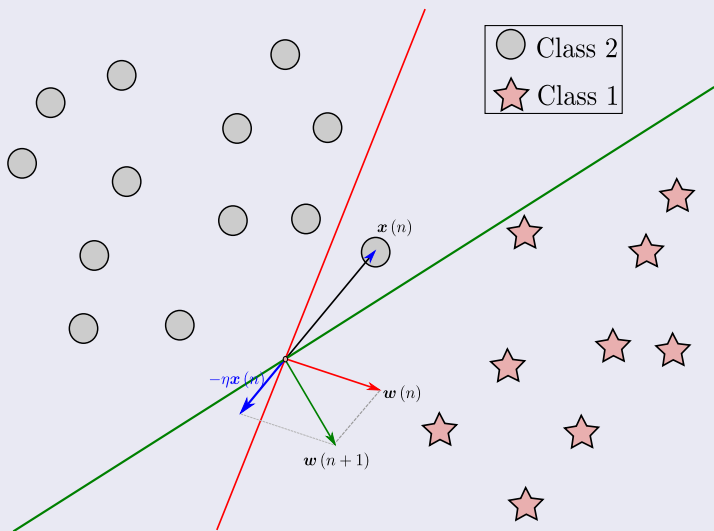
- 1  $\mathbf{w}(n+1) = \mathbf{w}(n) - \eta(n) \mathbf{x}(n)$  if  $\mathbf{w}^T(n) \mathbf{x}(n) > 0$  and  $\mathbf{x}(n)$  belongs to class  $C_2$ .
- 2  $\mathbf{w}(n+1) = \mathbf{w}(n) + \eta(n) \mathbf{x}(n)$  if and  $\mathbf{w}^T(n) \mathbf{x}(n) \leq 0$  and  $\mathbf{x}(n)$  belongs to class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

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



## A little bit on the Geometry

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



# Outline

## 1 Introduction

- History

## 2 Adapting Filtering Problem

- Definition
- Description of the Behavior of the System

## 3 Unconstrained Optimization

- Introduction
- Method of Steepest Descent
- Newton's Method
- Gauss-Newton Method

## 4 Linear Least-Squares Filter

- Introduction
- Least-Mean-Square (LMS) Algorithm

## 5 Perceptron

- Objective
- Perceptron: Local Field of a Neuron
- Perceptron: One Neuron Structure
- Deriving the Algorithm
- **Under Linear Separability - Convergence happens!!!**
- Proof
- Algorithm Using Error-Correcting
- Final Perceptron Algorithm (One Version)
- The Winnow Algorithm
- Parallel Implementations



# Under Linear Separability - Convergence happens!!!

If we assume

Linear Separability 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

$$w(n_0) = w(n_0 + 1) = w(n_0 + 2) = \dots \quad (33)$$

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



# Under Linear Separability - Convergence happens!!!

If we assume

Linear Separability 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

$$\mathbf{w}(n_0) = \mathbf{w}(n_0 + 1) = \mathbf{w}(n_0 + 2) = \dots \quad (33)$$

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





# Outline

## 1 Introduction

- History

## 2 Adapting Filtering Problem

- Definition
- Description of the Behavior of the System

## 3 Unconstrained Optimization

- Introduction
- Method of Steepest Descent
- Newton's Method
- Gauss-Newton Method

## 4 Linear Least-Squares Filter

- Introduction
- Least-Mean-Square (LMS) Algorithm

## 5 Perceptron

- Objective
- Perceptron: Local Field of a Neuron
- Perceptron: One Neuron Structure
- Deriving the Algorithm
- Under Linear Separability - Convergence happens!!!
- **Proof**
- Algorithm Using Error-Correcting
- Final Perceptron Algorithm (One Version)
- The Winnow Algorithm
- Parallel Implementations



# Proof I

## Initialization

$$\mathbf{w}(0) = \mathbf{0} \quad (34)$$

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

$$\mathbf{w}^T(n) \mathbf{x}(n) < 0 \quad (35)$$

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

PERCEPTRON INCORRECTLY CLASSIFY THE VECTORS  
 $\mathbf{x}(1), \mathbf{x}(2), \dots$

Apply the correction formula

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mathbf{x}(n) \quad (36)$$

# Proof I

## Initialization

$$\mathbf{w}(0) = 0 \quad (34)$$

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

$$\mathbf{w}^T(n) \mathbf{x}(n) < 0 \quad (35)$$

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

PERCEPTRON INCORRECTLY CLASSIFY THE VECTORS  
 $\mathbf{x}(1), \mathbf{x}(2), \dots$

Apply the correction formula

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mathbf{x}(n) \quad (36)$$

# Proof I

## Initialization

$$\mathbf{w}(0) = 0 \quad (34)$$

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

$$\mathbf{w}^T(n) \mathbf{x}(n) < 0 \quad (35)$$

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

PERCEPTRON INCORRECTLY CLASSIFY THE VECTORS  
 $\mathbf{x}(1), \mathbf{x}(2), \dots$

Apply the correction formula

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mathbf{x}(n) \quad (36)$$

## Proof II

Apply the correction iteratively

$$\mathbf{w}(n+1) = \mathbf{x}(1) + \mathbf{x}(2) + \dots + \mathbf{x}(n) \quad (37)$$

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

$$\alpha = \min_{\mathbf{x}(n) \in C_1} \mathbf{w}_0^T \mathbf{x}(n) \quad (38)$$

Then, we have

$$\mathbf{w}_0^T \mathbf{w}(n+1) = \mathbf{w}_0^T \mathbf{x}(1) + \mathbf{w}_0^T \mathbf{x}(2) + \dots + \mathbf{w}_0^T \mathbf{x}(n) \quad (39)$$



## Proof II

Apply the correction iteratively

$$\mathbf{w}(n+1) = \mathbf{x}(1) + \mathbf{x}(2) + \dots + \mathbf{x}(n) \quad (37)$$

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

$$\alpha = \min_{\mathbf{x}(n) \in C_1} \mathbf{w}_0^T \mathbf{x}(n) \quad (38)$$

Then, we have

$$\mathbf{w}_0^T \mathbf{w}(n+1) = \mathbf{w}_0^T \mathbf{x}(1) + \mathbf{w}_0^T \mathbf{x}(2) + \dots + \mathbf{w}_0^T \mathbf{x}(n) \quad (39)$$



## Proof II

Apply the correction iteratively

$$\mathbf{w}(n+1) = \mathbf{x}(1) + \mathbf{x}(2) + \dots + \mathbf{x}(n) \quad (37)$$

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

$$\alpha = \min_{\mathbf{x}(n) \in C_1} \mathbf{w}_0^T \mathbf{x}(n) \quad (38)$$

Then, we have

$$\mathbf{w}_0^T \mathbf{w}(n+1) = \mathbf{w}_0^T \mathbf{x}(1) + \mathbf{w}_0^T \mathbf{x}(2) + \dots + \mathbf{w}_0^T \mathbf{x}(n) \quad (39)$$



## Proof III

Apply the correction iteratively

$$\mathbf{w}(n+1) = \mathbf{x}(1) + \mathbf{x}(2) + \dots + \mathbf{x}(n) \quad (40)$$

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

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

Then, we have

$$\mathbf{w}_0^T \mathbf{w}(n+1) = \mathbf{w}_0^T \mathbf{x}(1) + \mathbf{w}_0^T \mathbf{x}(2) + \dots + \mathbf{w}_0^T \mathbf{x}(n) \quad (42)$$





## Proof III

Apply the correction iteratively

$$\mathbf{w}(n+1) = \mathbf{x}(1) + \mathbf{x}(2) + \dots + \mathbf{x}(n) \quad (40)$$

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

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

Then, we have

$$\mathbf{w}_0^T \mathbf{w}(n+1) = \mathbf{w}_0^T \mathbf{x}(1) + \mathbf{w}_0^T \mathbf{x}(2) + \dots + \mathbf{w}_0^T \mathbf{x}(n) \quad (42)$$



## Proof III

Apply the correction iteratively

$$\mathbf{w}(n+1) = \mathbf{x}(1) + \mathbf{x}(2) + \dots + \mathbf{x}(n) \quad (40)$$

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

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

Then, we have

$$\mathbf{w}_0^T \mathbf{w}(n+1) = \mathbf{w}_0^T \mathbf{x}(1) + \mathbf{w}_0^T \mathbf{x}(2) + \dots + \mathbf{w}_0^T \mathbf{x}(n) \quad (42)$$



## Proof IV

Thus we use the  $\alpha$

$$\mathbf{w}_0^T \mathbf{w}(n+1) \geq n\alpha \quad (43)$$

Thus using the Cauchy-Schwartz inequality

$$\|\mathbf{w}_0^T\|^2 \|\mathbf{w}(n+1)\|^2 \geq [\mathbf{w}_0^T \mathbf{w}(n+1)]^2 \quad (44)$$

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

This

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

## Proof IV

Thus we use the  $\alpha$

$$\mathbf{w}_0^T \mathbf{w}(n+1) \geq n\alpha \quad (43)$$

Thus using the Cauchy-Schwartz Inequality

$$\|\mathbf{w}_0^T\|^2 \|\mathbf{w}(n+1)\|^2 \geq [\mathbf{w}_0^T \mathbf{w}(n+1)]^2 \quad (44)$$

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

This

$$\|\mathbf{w}_0^T\|^2 \|\mathbf{w}(n+1)\|^2 \geq n^2 \alpha^2$$
$$\|\mathbf{w}(n+1)\|^2 \geq \frac{n^2 \alpha^2}{\|\mathbf{w}_0^T\|^2}$$

## Proof IV

Thus we use the  $\alpha$

$$\mathbf{w}_0^T \mathbf{w}(n+1) \geq n\alpha \quad (43)$$

Thus using the Cauchy-Schwartz Inequality

$$\|\mathbf{w}_0^T\|^2 \|\mathbf{w}(n+1)\|^2 \geq [\mathbf{w}_0^T \mathbf{w}(n+1)]^2 \quad (44)$$

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

Thus

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

## Proof V

Now rewrite equation 36

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \mathbf{x}(k) \quad (45)$$

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

Squaring the Euclidean norm of both sides

$$\|\mathbf{w}(k+1)\|^2 = \|\mathbf{w}(k)\|^2 + \|\mathbf{x}(k)\|^2 + 2\mathbf{w}^T(k)\mathbf{x}(k) \quad (46)$$

Now taking that we're in  $C_1$

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

## Proof V

Now rewrite equation 36

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \mathbf{x}(k) \quad (45)$$

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

Squaring the Euclidean norm of both sides

$$\|\mathbf{w}(k+1)\|^2 = \|\mathbf{w}(k)\|^2 + \|\mathbf{x}(k)\|^2 + 2\mathbf{w}^T(k)\mathbf{x}(k) \quad (46)$$

Now taking that we know  $\mathbf{x}(k) \in C_1$

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

## Proof V

Now rewrite equation 36

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \mathbf{x}(k) \quad (45)$$

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

Squaring the Euclidean norm of both sides

$$\|\mathbf{w}(k+1)\|^2 = \|\mathbf{w}(k)\|^2 + \|\mathbf{x}(k)\|^2 + 2\mathbf{w}^T(k)\mathbf{x}(k) \quad (46)$$

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

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



# Proof VI

Use the telescopic sum

$$\sum_{k=0}^n \left[ \|\mathbf{w}(k+1)\|^2 - \|\mathbf{w}(k)\|^2 \right] \leq \sum_{k=0}^n \|\mathbf{x}(k)\|^2 \quad (47)$$

Assume

$$\mathbf{w}(0) = \mathbf{0}$$

$$\mathbf{x}(0) = \mathbf{0}$$

Thus

$$\|\mathbf{w}(n+1)\|^2 \leq \sum_{k=0}^n \|\mathbf{x}(k)\|^2$$

## Proof VI

Use the telescopic sum

$$\sum_{k=0}^n \left[ \|\mathbf{w}(k+1)\|^2 - \|\mathbf{w}(k)\|^2 \right] \leq \sum_{k=0}^n \|\mathbf{x}(k)\|^2 \quad (47)$$

Assume

$$\mathbf{w}(0) = \mathbf{0}$$

$$\mathbf{x}(0) = \mathbf{0}$$

This

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

## Proof VI

Use the telescopic sum

$$\sum_{k=0}^n \left[ \|\mathbf{w}(k+1)\|^2 - \|\mathbf{w}(k)\|^2 \right] \leq \sum_{k=0}^n \|\mathbf{x}(k)\|^2 \quad (47)$$

Assume

$$\mathbf{w}(0) = \mathbf{0}$$

$$\mathbf{x}(0) = \mathbf{0}$$

Thus

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

## Proof VII

Then, we can define a positive number

$$\beta = \max_{\mathbf{x}(k) \in C_1} \|\mathbf{x}(k)\|^2 \quad (48)$$

This

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

Thus, we satisfies the equations only when exists a  $n$

$$\frac{n_{max}^2 \alpha^2}{\|w_0\|^2} = n_{max} \beta \quad (49)$$



## Proof VII

Then, we can define a positive number

$$\beta = \max_{\mathbf{x}(k) \in C_1} \|\mathbf{x}(k)\|^2 \quad (48)$$

Thus

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

Thus, we satisfies the equalities only when exists a  $\mathbf{x}(k)$

$$\frac{n_{max}^2 \alpha^2}{\|\mathbf{w}_0\|^2} = n_{max} \beta \quad (49)$$



## Proof VII

Then, we can define a positive number

$$\beta = \max_{\mathbf{x}(k) \in C_1} \|\mathbf{x}(k)\|^2 \quad (48)$$

Thus

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

Thus, we satisfies the equations only when exists a  $n_{max}$

$$\frac{n_{max}^2 \alpha^2}{\|\mathbf{w}_0\|^2} = n_{max} \beta \quad (49)$$



# Proof VIII

## Solving

$$n_{max} = \frac{\beta \|w_0\|^2}{\alpha^2} \quad (50)$$

## Thus

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

- The rule for adapting 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

$$n_{max} = \frac{\beta \|w_0\|^2}{\alpha^2} \quad (50)$$

### Thus

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

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

### Conclusion

Because  $w_0$  the solution is not unique.





## Proof VIII

### Solving

$$n_{max} = \frac{\beta \|w_0\|^2}{\alpha^2} \quad (50)$$

### Thus

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

- The rule for adapting 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

## 1 Introduction

- History

## 2 Adapting Filtering Problem

- Definition
- Description of the Behavior of the System

## 3 Unconstrained Optimization

- Introduction
- Method of Steepest Descent
- Newton's Method
- Gauss-Newton Method

## 4 Linear Least-Squares Filter

- Introduction
- Least-Mean-Square (LMS) Algorithm

## 5 Perceptron

- Objective
- Perceptron: Local Field of a Neuron
- Perceptron: One Neuron Structure
- Deriving the Algorithm
- Under Linear Separability - Convergence happens!!!
- Proof
- **Algorithm Using Error-Correcting**
- Final Perceptron Algorithm (One Version)
- The Winnow Algorithm
- Parallel Implementations



# 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

$$\Delta w(n) = \eta((d_j - y_j(n))) x(n) \quad (51)$$



# 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

$$\Delta \mathbf{w}(n) = \eta ((d_j - y_j(n))) \mathbf{x}(n) \quad (51)$$



# Outline

## 1 Introduction

- History

## 2 Adapting Filtering Problem

- Definition
- Description of the Behavior of the System

## 3 Unconstrained Optimization

- Introduction
- Method of Steepest Descent
- Newton's Method
- Gauss-Newton Method

## 4 Linear Least-Squares Filter

- Introduction
- Least-Mean-Square (LMS) Algorithm

## 5 Perceptron

- Objective
- Perceptron: Local Field of a Neuron
- Perceptron: One Neuron Structure
- Deriving the Algorithm
- Under Linear Separability - Convergence happens!!!
- Proof
- Algorithm Using Error-Correcting
- **Final Perceptron Algorithm (One Version)**
- The Winnow Algorithm
- Parallel Implementations



# Final Algorithm

## Algorithm - Offline 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  $j = 1, 2, \dots, m$ .
- 3 Initialize the weights,  $w^T = (w_1(n), w_2(n), \dots, w_n(n))$ .
  - ▶ Weights may be initialized to 0 or to a small random value.
- 4 Initialize Dummy outputs so you can enter loop  $y^t = (y_1(n), y_2(n), \dots, y_m(n))$
- 5 Initialize Stopping error  $\epsilon > 0$ .
- 6 Initialize learning error  $\eta$ .
- 7 While  $\frac{1}{m} \sum_{j=1}^m \|d_j - y_j(n)\| > \epsilon$ 
  - ▶ For each sample  $(x_j, d_j)$  for  $j = 1, \dots, m$ :
    - ★ Calculate output  $y_j = \varphi(w^T(n) \cdot x_j)$
    - ★ Update weights  $w_i(n+1) = w_i(n) + \eta(d_j - y_j(n))x_{ij}$ .
  - ▶  $n = n + 1$

# Final Algorithm

## Algorithm - Offline/Batch Learning

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

# Outline

## 1 Introduction

- History

## 2 Adapting Filtering Problem

- Definition
- Description of the Behavior of the System

## 3 Unconstrained Optimization

- Introduction
- Method of Steepest Descent
- Newton's Method
- Gauss-Newton Method

## 4 Linear Least-Squares Filter

- Introduction
- Least-Mean-Square (LMS) Algorithm

## 5 Perceptron

- Objective
- Perceptron: Local Field of a Neuron
- Perceptron: One Neuron Structure
- Deriving the Algorithm
- Under Linear Separability - Convergence happens!!!
- Proof
- Algorithm Using Error-Correcting
- Final Perceptron Algorithm (One Version)
- **The Winnow Algorithm**
- Parallel Implementations





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 $w$

- $w^k = (w_1, w_2, \dots, w_p)$  for all  $w_i \in \mathbb{R}$
- 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 $w$

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



# Classification Scheme

We use a specific  $\theta$

- 1  $w^T x \geq \theta \Rightarrow$  positive classification Class 1
- 2  $w^T x < \theta \Rightarrow$  positive classification 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  $x \in \text{Class 1}$  i.e.  $w^T x < \theta$

$$\forall x_i = 1 : w_i \leftarrow \alpha w_i$$

(52)



Cinvestav

# Classification Scheme

We use a specific  $\theta$

- 1  $w^T x \geq \theta \Rightarrow$  positive classification Class 1
- 2  $w^T x < \theta \Rightarrow$  positive classification Class 2

## Rule

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

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

$$\forall x_i = 1 : w_i \leftarrow \alpha w_i$$

(52)



Cinvestav

# Classification Scheme

We use a specific  $\theta$

- 1  $w^T x \geq \theta \Rightarrow$  positive classification Class 1
- 2  $w^T x < \theta \Rightarrow$  positive classification 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  $x \in \text{Class 1}$  i.e.  $w^T x < \theta$

$$\forall x_i = 1 : w_i \leftarrow \alpha w_i \quad (52)$$



# Classification Scheme

## Rule 2

- When misclassifying a negative training example  $x \in \text{Class 1}$  i.e.  
 $w^T x \geq \theta$

$$\forall x_i = 1 : w_i \leftarrow \frac{w_i}{\alpha} \quad (53)$$

## Rule 3

- If samples are correctly classified do nothing!!!



# Classification Scheme

## Rule 2

- When misclassifying a negative training example  $x \in \text{Class 1}$  i.e.  
 $w^T x \geq \theta$

$$\forall x_i = 1 : w_i \leftarrow \frac{w_i}{\alpha} \quad (53)$$

## Rule 3

- If samples are correctly classified do nothing!!!





# Properties of Winnow

## Property

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

## Drawback

- Sensitive to the learning rate  $\alpha$ .



# Properties of Winnow

## Property

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

## Drawback

- Sensitive to the learning rate  $\alpha$ .



# Outline

## 1 Introduction

- History

## 2 Adapting Filtering Problem

- Definition
- Description of the Behavior of the System

## 3 Unconstrained Optimization

- Introduction
- Method of Steepest Descent
- Newton's Method
- Gauss-Newton Method

## 4 Linear Least-Squares Filter

- Introduction
- Least-Mean-Square (LMS) Algorithm

## 5 Perceptron

- Objective
- Perceptron: Local Field of a Neuron
- Perceptron: One Neuron Structure
- Deriving the Algorithm
- Under Linear Separability - Convergence happens!!!
- Proof
- Algorithm Using Error-Correcting
- Final Perceptron Algorithm (One Version)
- The Winnow Algorithm
- **Parallel Implementations**



# Parallel Implementations

Because for the Perceptron, we have this

## Iterative Parameter Mixing

### Parameter Mixing

It is a distributed training through parameter mixing is a straight-forward way of training classifiers in parallel.

### Basic Idea

Take the data set  $\{x_i, y_i\}_{i=1}^N$ , then:

- Split it into  $S$  disjoint shards  $T = \{T_1, T_2, \dots, T_S\}$ .
- Train each parallel machine using one of the shards.
- After Training, you get the set of parameters  $\{\theta_1, \theta_2, \dots, \theta_S\}$ 
  - Obtain  $\hat{\theta} = \sum_{i=1}^S \mu_i \theta_i$

# Parallel Implementations

Because for the Perceptron, we have this

## Iterative Parameter Mixing

### Parameter Mixing

It is a distributed training through parameter mixing is a straight-forward way of training classifiers in parallel.

#### Basic Idea

Take the data set  $\{x_i, y_i\}_{i=1}^N$ , then:

- Split it into  $S$  disjoint shards  $T = \{T_1, T_2, \dots, T_S\}$ .
- Train each parallel machine using one of the shards.
- After Training, you get the set of parameters  $\{\Theta_1, \Theta_2, \dots, \Theta_S\}$ .
- Obtain  $\tilde{\Theta} = \sum_{i=1}^S \mu_i \Theta_i$ .

# Parallel Implementations

Because for the Perceptron, we have this

## Iterative Parameter Mixing

### Parameter Mixing

It is a distributed training through parameter mixing is a straight-forward way of training classifiers in parallel.

### Basic Ideas

Take the data set  $\{\mathbf{x}_i, y_i\}_{i=1}^N$ , then:

- 1 Split it into  $S$  disjoint shards  $T = \{T_1, T_2, \dots, T_S\}$ .
- 2 Train each parallel machine using one of the shards.
- 3 After Training, you get the set of parameters  $\{\Theta_1, \Theta_2, \dots, \Theta_S\}$ 
  - 1 Obtain  $\hat{\Theta} = \sum_{i=1}^S \mu_i \Theta_i$

# What about the code for the Perceptron?

PerceptronIterParamMix( $T = \{(\mathbf{x}_t, y_t)\}_{t=1}^{|T|}$ )

- 1 Shard  $T$  into  $S$  pieces  $T = \{T_1, \dots, T_S\}$
- 2  $\mathbf{w} = 0$
- 3 # Here the parallel epochs!!!
- 4 for  $j = 1$  to  $N$
- 5     Parallel computation Part for  $i = 1, \dots, S$
- 6          $\mathbf{w}^{(i,j)} = \text{OneEpochPerceptron}(T_i, \mathbf{w})$
- 7          $\mathbf{w} = \sum_i \mu_{ij} \mathbf{w}^{(i,j)}$
- 8 return  $\mathbf{w}$

Here, we have for the  $\mu_{ij}$ s

For  $\mu_j = \{\mu_{1j}, \mu_{2j}, \dots, \mu_{Sj}\}$ ,  $\mu_{ij} \geq 0$  and  $\forall j \sum_i \mu_{ij} = 1$ .

## What about the code for the Perceptron?

PerceptronIterParamMix( $T = \{(\mathbf{x}_t, y_t)\}_{t=1}^{|T|}$ )

- 1 Shard  $T$  into  $S$  pieces  $T = \{T_1, \dots, T_S\}$
- 2  $\mathbf{w} = 0$
- 3 # Here the parallel epochs!!!
- 4 for  $j = 1$  to  $N$
- 5     Parallel computation Part for  $i = 1, \dots, S$
- 6          $\mathbf{w}^{(i,j)} = \text{OneEpochPerceptron}(T_i, \mathbf{w})$
- 7          $\mathbf{w} = \sum_i \mu_{ij} \mathbf{w}^{(i,j)}$
- 8 return  $\mathbf{w}$

Here, we have for the  $\mu$ 's

For  $\boldsymbol{\mu}_j = \{\mu_{1j}, \mu_{2j}, \dots, \mu_{Sj}\}$ ,  $\mu_{ij} \geq 0$  and  $\forall j \sum_i \mu_{ij} = 1$ .



# What about the code for the Perceptron?

## OneEpochPerceptron( $T_i, w^*$ )

- 1  $w_0 = w^*; k = 0$
- 2  $w = 0$
- 3 for  $t = 1$  to  $|T|$
- 4     Let  $y' = \arg \max_y \{w_k^t f(x_t, y)\}$
- 5      $w_{(k+1)} = w_{(k)} + [f(x_t, y_t) - f(x_t, y')]$
- 6      $k = k + 1$
- 7 return  $w$

Where / how?

- It is the high dimensional representation of the pair  $(x_t, y)$ .



# What about the code for the Perceptron?






## OneEpochPerceptron( $T_i, w^*$ )

- 1  $w_0 = w^*; k = 0$
- 2  $w = 0$
- 3 for  $t = 1$  to  $|T|$
- 4     Let  $y' = \arg \max_y \{w_k^t f(x_t, y)\}$
- 5      $w_{(k+1)} = w_{(k)} + [f(x_t, y_t) - f(x_t, y')]$
- 6      $k = k + 1$
- 7 return  $w$

## Where $f(x_t, y)$

- It is the high dimensional representation of the pair  $(x_t, y)$ .



-  W. S. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.
-  D. O. Hebb, *The organization of behavior: a neuropsychological theory*.  
Science Editions, 1962.
-  F. Rosenblatt, “The perceptron: a probabilistic model for information storage and organization in the brain.,” *Psychological review*, vol. 65, no. 6, p. 386, 1958.
-  H. B. Curry, “The method of steepest descent for non-linear minimization problems,” *Quarterly of Applied Mathematics*, vol. 2, no. 3, pp. 258–261, 1944.
-  S. S. Haykin, *Adaptive filter theory*.  
Pearson Education India, 2005.

