Fundamentos de aprendizaje automático
Clasificadores basados en vectores soportes
Segunda parte

Juan Miguel Santos
Centro de investigación y desarrollo en informática aplicada (CIDIA)
Universidad Nacional de Hurlingham
2023

Diferencias entre procesamiento máquina de Von Newman y sistemas biológicos

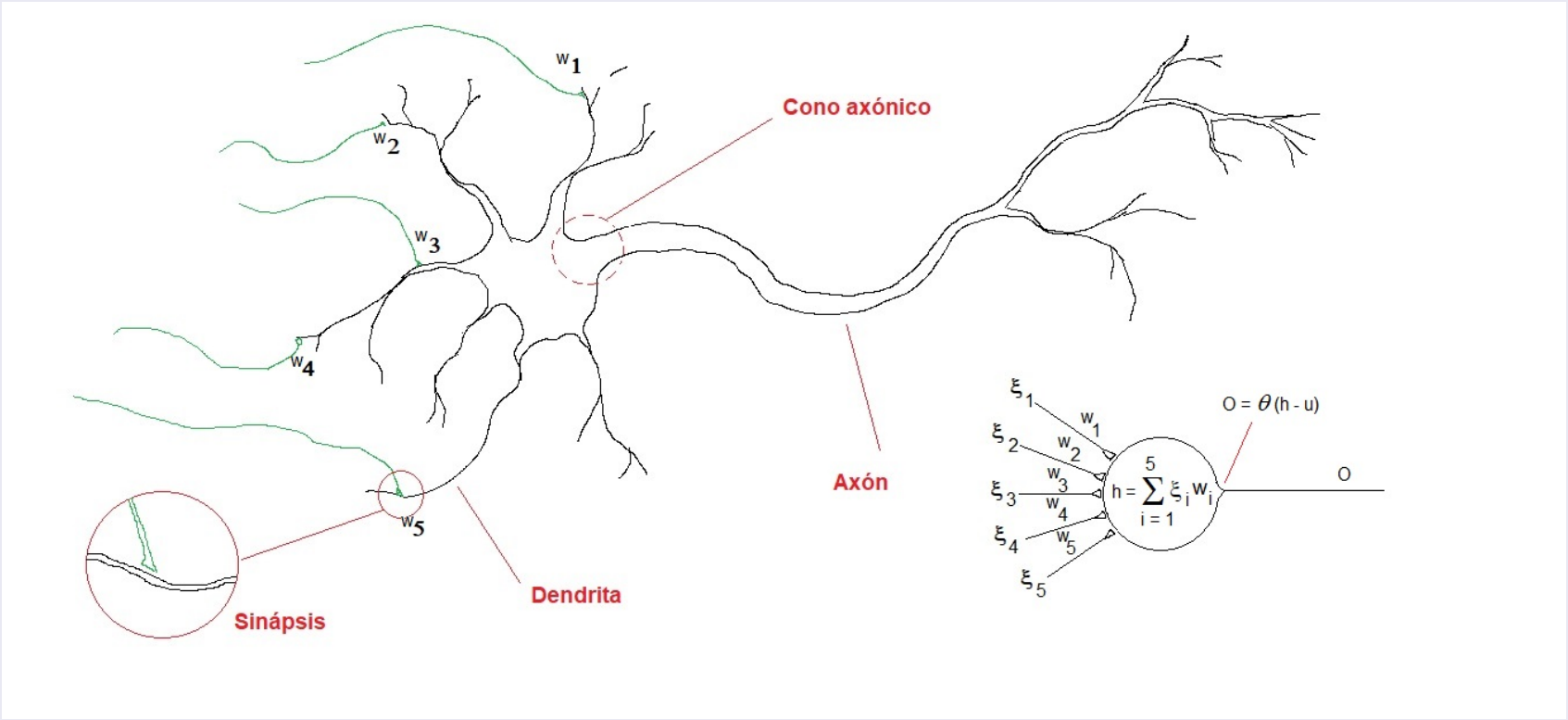
- Uno o unos pocos procesadores con gran poder de procesamiento.
  Muchísimos (miles de millones) procesadores con procesamiento 'simple'.
- Escasas conexiones estables entre procesadores, mediadas por buses o por acceso a memoria compartida.
  Conexión del tipo masiva (una neurona puede estar conectada hasta con otras 10000 neuronas) que puede ser cambiantes en el tiempo.

Diferencias entre procesamiento máquina de Von Newman y sistemas biológicos

- Tiempos de conmutación y transmisión de información muy breves.
  Los tiempos de conmutación y transmisión son parte de la dinámica del sistema.
- Memoria localizada en dispositivos de almacenamiento.
  Memoria distribuida en la estructura de la red neuronal (conjetura de Hebb como una posible explicación).
- Reducida tolerancia a fallas si una parte es dañada.
  Pueden re-estructurarse en caso de daños.
- Máquinas que se programan.
  Aprendizaje por ejemplos.

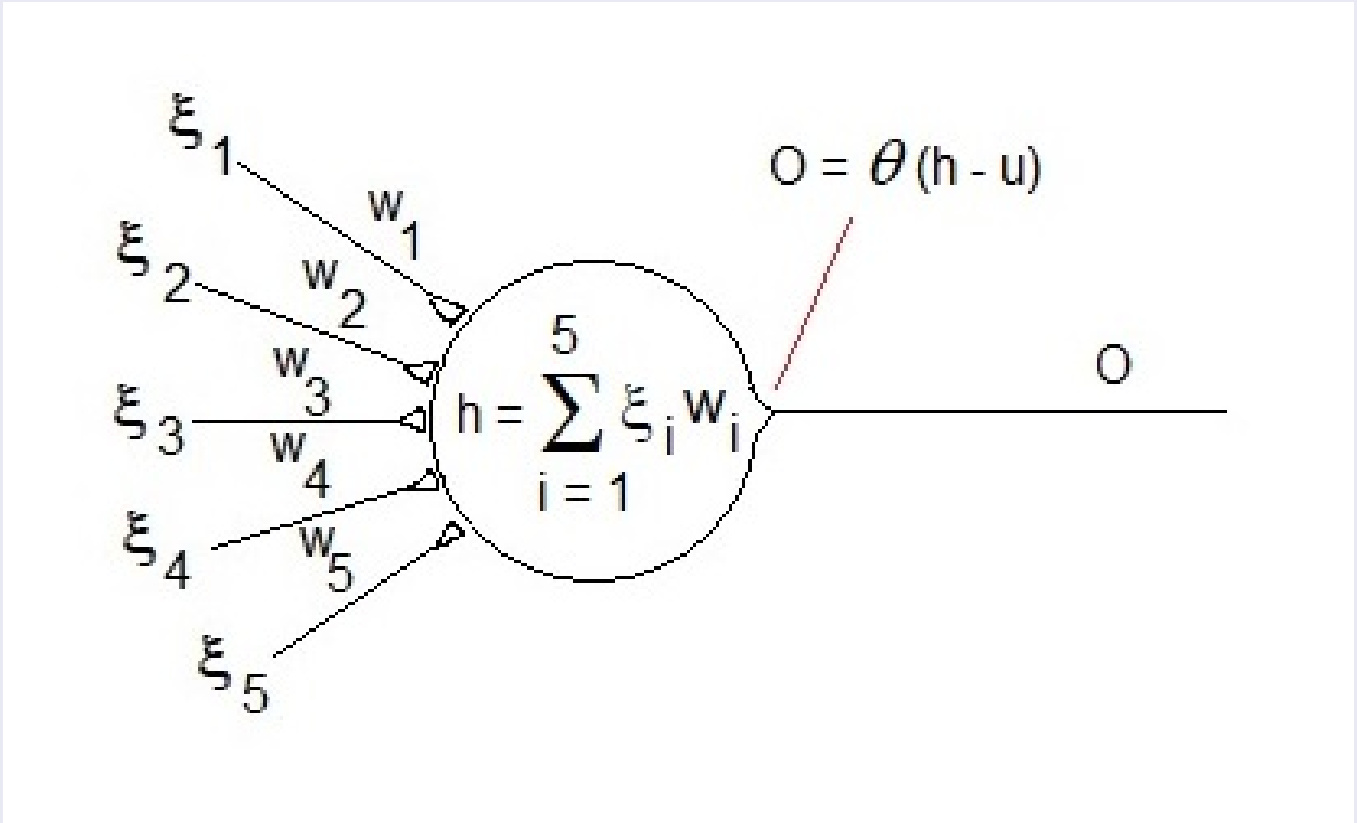
Perceptron simple. Modelo de neurona

El esquema de neurona en que se basaron fue:



Redes neuronales artificiales

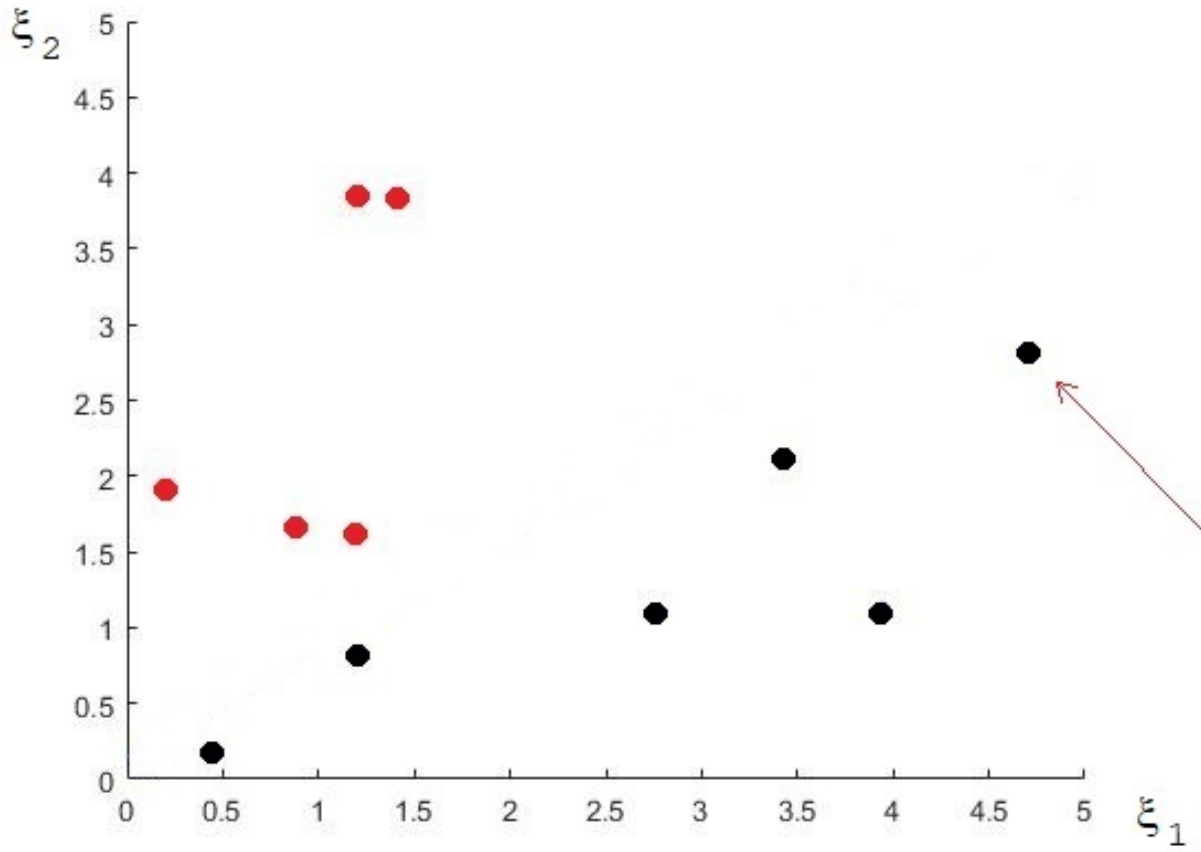
El modelo de neurona que propusieron fue:



Perceptron simple unidad escalón

- Simplificación de la estructura de una neurona.
- Entrada (usualmente notada como ξ ó x).
- Exitación e inhibición. Pesos sinápticos (usualmente notados como w).
- Estado de exitación (usualmente notado como h).
- Estado de activación (usualmente notado como O).
- Función de activación (función escalón con imagen en $\{-1, 1\}$ o $\{0, 1\}$).
- Umbral (usualmente notado como u ó w_0).

Perceptron simple. Un problema para resolver ....



ξ_1	ξ_2	ζ
1.1946	3.8427	1
0.8788	1.6595	1
1.1907	1.6117	1
1.4180	3.8272	1
0.2032	1.9208	1
2.7571	1.0931	-1
4.7125	2.8166	-1
3.9392	1.1032	-1
1.2072	0.8132	-1
3.4799	1.9982	-1
0.4763	0.1020	-1

Regla de actualización perceptron simple

- Δw $= \eta$ * Error * x
  donde el Error = salida deseada - salida real, y
  x es un patrón de entrenamiento

Algoritmo perceptron simple

Pseudocódigo

$i \leftarrow 0$
Inicializar w en ceros;
$error \leftarrow 1$
$error\_min \leftarrow p * 2$
while $error > 0 \wedge i < COTA$
    Elegir al azar un ejemplo x' siendo su salida deseada y'
    $exitacion \leftarrow x' * w$
    $activacion \leftarrow signo(exitacion)$
    $\Delta w \leftarrow \eta * (y' - activacion) * x'$
    $w \leftarrow w + \Delta w$;
    error = CalcularError(x, y, w, p);
    if $error < error\_min$
        $error\_min \leftarrow error$
        $w\_min \leftarrow w$
    end
    $i \leftarrow i + 1$
end

Algoritmo perceptron simple

- Una época es cuando fueron expuestos al perceptrón todos los ejemplos del conjunto de entrenamiento.
- p es la cantidad de ejemplos del conjunto de entrenamiento.
- x es el conjunto de entrenamiento donde a cada ejemplo se le agrega una componente $= 1$ (aumenta en 1 la dimensión donde se representa el ejemplo).
- y es la salida deseada (clase).
- w es el vector de pesos 'sinápticos' que incluye el umbral (el w tiene la misma dimensión que los ejemplos - con el '1' incluído).
- *signo*() es la función de activación.

Algoritmo perceptron simple

- Mostrar implementación (sólo parte de perceptron) ...

Algoritmo perceptron simple para hiperplano óptimo

- Mostrar implementación (determinación de hiperplano óptimo) ...

¿Cómo encontrar el hiperplano óptimo?

- Fijarse en el puntos alrededor del hiperplano
- Buscar las combinaciones de candidatos a vectores de soporte
- Evaluar los hiperplanos obtenidos y conservar el que maximiza el margen.

Algunos apuntes de como usar librerías de SVM

C++ y Python

```
#include ⟨ iostream⟩
#include ⟨ opencv2/core.hpp⟩
#include ⟨ opencv2/imgproc.hpp⟩
#include ⟨ opencv2/imgcodecs.hpp⟩
#include ⟨ opencv2/highgui.hpp⟩
#include ⟨ opencv2/ml.hpp⟩
using namespace cv;
using namespace cv::ml;
using namespace std;
```

Algunos apuntes de como usar librerías de SVM

C++ y Python

- Inicialización

```
Ptr⟨SVMe⟩ svm = SVM::create();
svm→setType(SVM::C_SVC);
svm→setC(0.1);
svm→setKernel(SVM::LINEAR);
svm→setTermCriteria(TermCriteria(
TermCriteria::MAX_ITER, (int)1e7, 1e-6));
```

Algunos apuntes de como usar librerías de SVM

C++ y Python

donde

- SetKernel() puede recibir como argumento LINEAR=0, POLY=1, RBF=2, ...,
- setType() puede recibir como argumento C_SVC donde la C es un factor de penalidad por *mal clasificación*,
- setTermCriteria() puede recibir un valor de tolerancia o una cantidad de iteraciones para resolver un caso parcial de la optimización cuadrática. El valor por defecto es TermCriteria(TermCriteria::MAX_ITER + TermCriteria::EPS, 1000, FLT_EPSILON)

Algunos apuntes de como usar librerías de SVM

C++ y Python

- Entrenamiento
  svm→**train**(trainData, ROW_SAMPLE, labels);
- Testeo
  Mat sv = svm→**getUncompressedSupportVectors**();
  float r = svm→**predict**(testeo); donde testeo es una observación de test.

Algunos apuntes de como usar librerías de SVM

MatLab, (R2016b)

SVM = **svmtrain**(X, Y)
donde

- las filas de X son los ejemplos,
- las columnas de X son los atributos,
- Y son las clases de los ejemplos de X
- SVM es una estructura que contiene información acerca del clasificador entrenado y los vectores de

SVM = **svmtrain**(X, Y, **'kernel_function'**, tipo)
donde tipo puede ser: 'linear', 'quadratic', 'polynomial', 'rbf', ...

Algunos apuntes de como usar librerías de SVM

MatLab, (R2016b)

SVM = **svmtrain**(X, Y, 'kernel_function', tipo, **'boxconstraint'**, C)
donde C puede ser un escalar positivo o un vector de valores positivos (de las misma dimensión que las filas de X).
SVM = **svmtrain**(X, Y, 'kernel_function', tipo, 'boxconstraint', C, **'showplot'**, mostrar)
donde mostrar es verdadero (true) o falso (false). Si la cantidad de columnas de X es distinto de 2, 'showplot' es siempre falso.

Algunos apuntes de como usar librerías de SVM

MatLab, (R2016b)

grupos = **svmclassify**(SVM, test) donde

- SVM es la estructura obtenida por svmtrain(), test es una matriz de observaciones para testeo,
- grupos (que tiene la misma cantidad de filas que test) es el valor predicho por el clasificador.

grupos = **svmclassify**(SVM, test, **'SHOWPLOT'**, true) clasifica y muestra el resultado de la clasificación.