|  |  |  |
| --- | --- | --- |
| **ipn** | **INSTITUTO POLITÉCNICO NACIONAL**  **ESCUELA SUPERIOR DE CÓMPUTO** |  |

**Neural Networks**

**“Perceptrón Multicapa”**

Resumen

Implementación en MATLAB de un perceptrón multicapa de múltiples neuronas para aproximar señales en un rango definido por el usuario y utilizando el algoritmo Early Stopping para evitar el sobre entrenamiento de la red y evitar resultados extraños.

**Por:**

**Joel Mauricio Romero Gamarra**

Profesor:

MORENO ARMENDÁRIZ MARCO ANTONIO

Noviembre 2017

**Índice**

Contenido

[Introducción: 1](#_Toc500807955)

[Análisis Teórico: 3](#_Toc500807956)

[Software (librarías, paquetes, herramientas): 4](#_Toc500807957)

[Resultados 5](#_Toc500807958)

[Conclusiones: 18](#_Toc500807959)

[Referencias: 19](#_Toc500807960)

[Código 19](#_Toc500807961)

# Introducción:

Un perceptrón multicapa puede resolver problemas que un perceptrón simple (que tiene una sola capa) no puede, un ejemplo muy simple es la clasificación de patrones con una simple compuerta XOR, de la cual, la tabla de verdad que representa su comportamiento se describe a continuación:

|  |  |  |
| --- | --- | --- |
| A | B | XOR |
| 0 | **0** | **0** |
| 0 | **1** | **1** |
| 1 | **0** | **1** |
| 1 | **1** | **0** |

Tabla 1. Tabla de verdad de compuerta XOR

Se puede ver en la Tabla 1, que existen 2 clases para esta compuerta (la clase 0 y la clase 1), que se puede modelar mediante una función de activación hardlim (ya que separa en las clases 0 y 1). Este problema fue usado por Minsky y Popert en 1969 para demostrar las limitaciones del perceptrón simple. [1]

Estas 2 clases no son linealmente separables y se puede apreciar en la Figura 1.

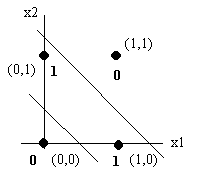


Figura . Gráfica compuerta XOR

Como podemos observar en la figura 1, se observan las 2 clases que se forman con la compuerta XOR, un perceptrón simple de 1 sola capa no es capaz de separarlas debido a que es un problema que no se linealmente separable, ya que primero se necesita dibujar las 2 fronteras de decisión ya dibujadas y posteriormente combinar ambas fronteras.

Para dibujar cada frontera de decisión, es necesario una neurona, es decir, que en la primera capa del perceptrón multicapa se necesitan 2 neuronas que dibujen ambas fronteras de decisión y en la capa 2 del MLP necesitamos una operación AND para combinar ambas fronteras.

El MLP tiene un algoritmo de aprendizaje tipo Back Propagation para actualizar los valores de pesos y bias en cada iteración y en cada capa. En esta práctica, se utiliza el perceptrón multicapa para aproximar señales dadas, leídas de un archivo de texto.

Recordando el modelo matemático o propagación hacia delante de un perceptrón multicapa, es:

a0 = p

am+1 = fm+1 (Wm+1 · am + bm+1), m = 0, 1, 2, …, M – 1

donde:

* M = Número de capas

Un perceptrón multicapa tiene como restricción que en todas sus capas ocultas debe contener funciones continuas y derivables, debido a las sensitividades para su algoritmo de aprendizaje, debido a que se deben derivar las funciones de activación correspondientes (por cada capa) y ponerlas en la diagonal principal de la matriz F. En la Figura 2 se puede ver un ejemplo general (es decir, no están definidas las funciones de activación) para un perceptrón multicapa con 2 capas ocultas y 1 capa exterior.

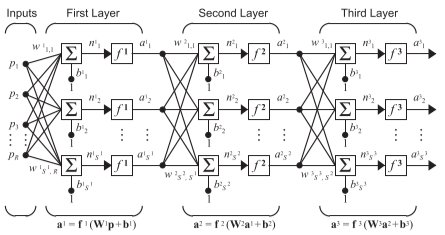


Figura . Arquitectura de un MLP de 3 capas (Generalizado) [2]

Como podemos ver, cada una de las capas tiene una función de activación distinta, a continuación, se muestra en la Figura 3 un ejemplo más particular de un MLP de 2 capas (1 capa oculta), con una función logsig (que es una función derivable) en su capa oculta y una capa lineal en la capa exterior (purelin).

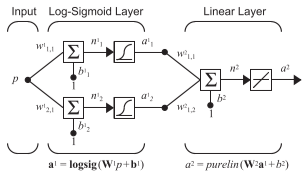


Figura . Arquitectura de un MLP de 2 capas [2]

En RNA’s pasadas, habíamos visto los criterios de finalización, como por ejemplo llegar hasta la iteración máxima introducida por el usuario o que el error de iteración sea menor al error de iteración que introdujo el usuario.

Sin embargo, en este tipo de red neuronal, existe un algoritmo que sirve para finalizar el programa y evita el sobre entrenamiento de la red neuronal mostrado en la Figura 4.

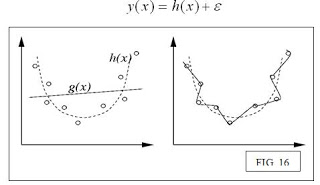


Figura . Sobre entrenamiento de una RNA

El sobre entrenamiento quiere decir que un error (introducido en el MLP) de validación tuvo n incrementos consecutivos, es decir, que ese tipo de error ya aumentó demasiado, esto provocaría un sobre entrenamiento que significa que al final puede quedar con datos muy extraños la RNA y en vez de aproximar correctamente, puede arrojar resultados demasiado disparados de lo que en verdad son.

En la siguiente sección, se explica a fondo el algoritmo de aprendizaje back Propagation que utiliza el MLP, junto con el concepto de **sensitividades**.

# Análisis Teórico:

En resumen, el algoritmo Back Propagation utiliza las siguientes ecuaciones: [1]

1. **SENSITIVIDADES**

SM = -2F’M (n) (t – a)

Sm = Fm (nm) (Wm + 1)T Sm + 1

Para m = 0, 1, 2, …, M – 1

La parte de F’m (nm) se refiere a la derivada de la función de activación en forma de una matriz de la siguiente forma:

Donde:

Las reglas de aprendizaje (actualización de los pesos y bias) son las siguientes:

Como ya se había visto, m va de 0 hasta M – 1, donde M es el número de capas del perceptrón multicapa. Un perceptrón puede tener M capas, con la única condición de que las capas ocultas deben ser no lineales y que la última capa o capa exterior, siempre debe ser purelin.

1. **PROPAGACIÓN HACIA ATRÁS**

Seguimos las fórmulas presentadas en la parte de arriba, ya que la capa exterior siempre debe ser purelin, para la sensitividad de la capa exterior, la respuesta siempre será 1.

Y para formar la matriz de F para calcular las sensitividades de las otras capas, las derivadas de las funciones de activación no lineales que se deben usar (tansig o logsig) son las siguientes:

* Logsig = 1 – (a21)2
* Tansig = a11 [1 – a11]

Una vez que tenemos todas las sensitividades podemos comenzar a calcular los valores de las matrices de pesos y los bias para comenzar la propagación hacia delante de los datos.

# Software (librarías, paquetes, herramientas):

* MATLAB R2016a [3]
* Notepad ++ [4]

# Resultados

* **01\_POLINOMIO**

La primera prueba para el programa es el polinomio representado por los siguientes valores:

-2.0000

-1.9600

-1.9200

-1.8800

-1.8400

-1.8000

-1.7600

-1.7200

-1.6800

-1.6400

-1.6000

-1.5600

-1.5200

-1.4800

-1.4400

-1.4000

-1.3600

-1.3200

-1.2800

-1.2400

-1.2000

-1.1600

-1.1200

-1.0800

-1.0400

-1.0000

-0.9600

-0.9200

-0.8800

-0.8400

-0.8000

-0.7600

-0.7200

-0.6800

-0.6400

-0.6000

-0.5600

-0.5200

-0.4800

-0.4400

-0.4000

-0.3600

-0.3200

-0.2800

-0.2400

-0.2000

-0.1600

-0.1200

-0.0800

-0.0400

0

0.0400

0.0800

0.1200

0.1600

0.2000

0.2400

0.2800

0.3200

0.3600

0.4000

0.4400

0.4800

0.5200

0.5600

0.6000

0.6400

0.6800

0.7200

0.7600

0.8000

0.8400

0.8800

0.9200

0.9600

1.0000

1.0400

1.0800

1.1200

1.1600

1.2000

1.2400

1.2800

1.3200

1.3600

1.4000

1.4400

1.4800

1.5200

1.5600

1.6000

1.6400

1.6800

1.7200

1.7600

1.8000

1.8400

1.8800

1.9200

1.9600

2.0000

En este caso, el conjunto de Targets tiene 101 datos, así que para esta prueba se utilizará la configuración 80 – 10 – 10.

El programa al momento de la ejecución pedirá los siguientes datos al usuario:

1. Nombre del archivo de entrada
2. Nombre del archivo con los targets
3. Rango de la señal a aproximar
4. Arquitectura del MLP en 2 vectores
5. Valor de α (Factor de aprendizaje)
6. Valor mínimo del error de entrenamiento
7. Cada cuántas iteración se realizará una iteración de validación
8. Número de incrementos consecutivos del error de validación
9. Configuración deseada (80 – 10 – 10 o 70 – 15 – 15)

Se puede apreciar a continuación en la Figuras 5, 6 y 7 los valores que ingresa el usuario y de qué forma los ingresa:

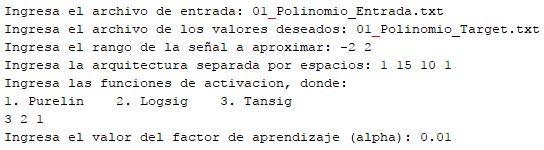


Figura . Valores ingresados por el usuario

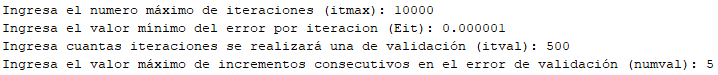


Figura . Criterios de finalización ingresados por el usuario

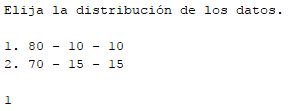


Figura . Configuración de la RNA

A continuación, se muestran los resultados obtenidos:

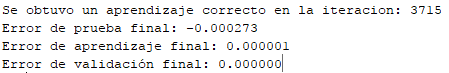


Figura 8. Aprendizaje correcto

A continuación, en las Figuras 9 a 17 se muestran las gráficas que presentan el resultado al usuario.

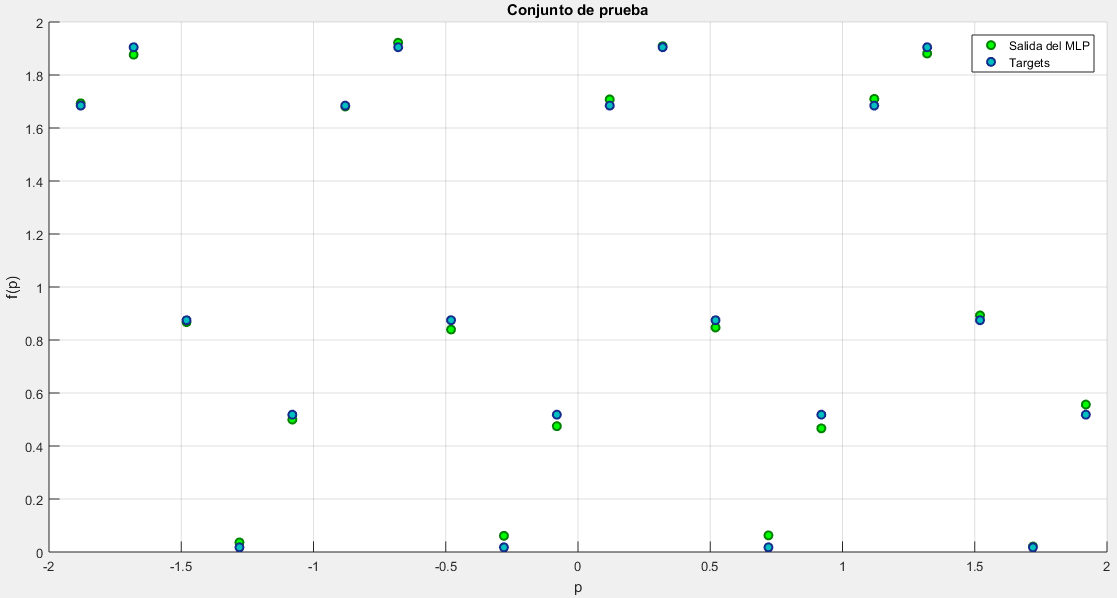


Figura . Subconjunto de prueba con sus targets

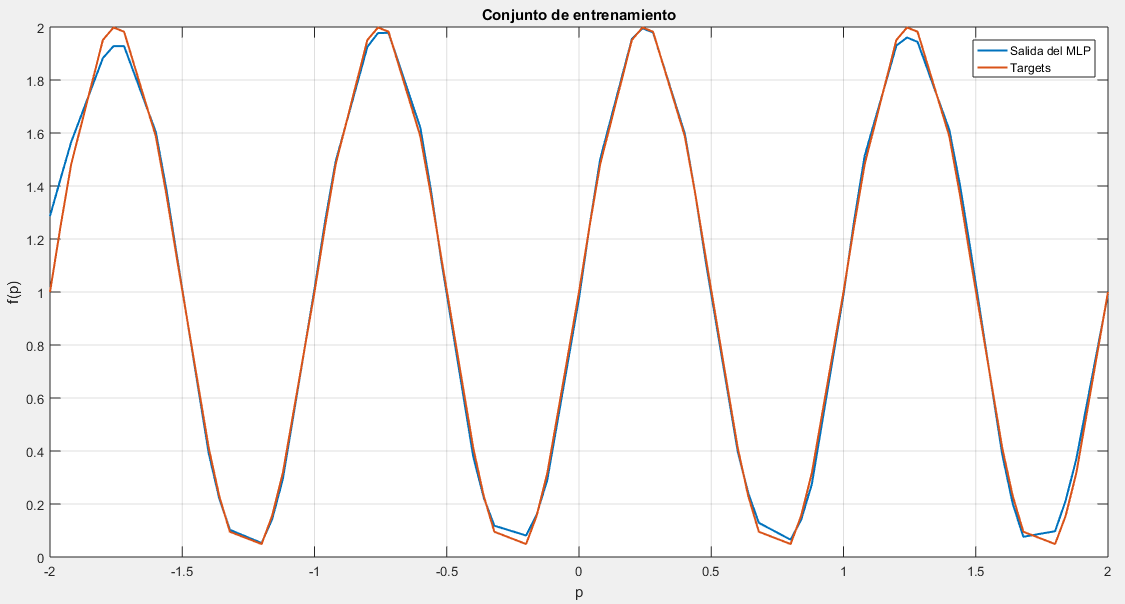


Figura . Subconjunto de entrenamiento con sus targets

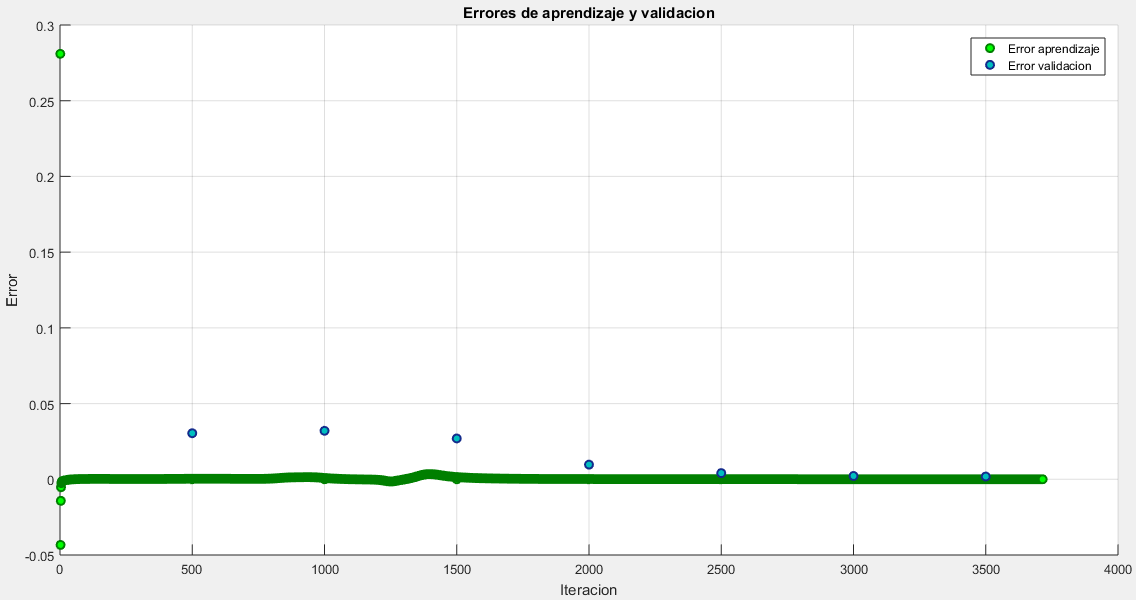


Figura . Errores de aprendizaje y de validación

Ahora, se muestra en las Figuras 12, 13 y 14 la evolución de los pesos por capa.

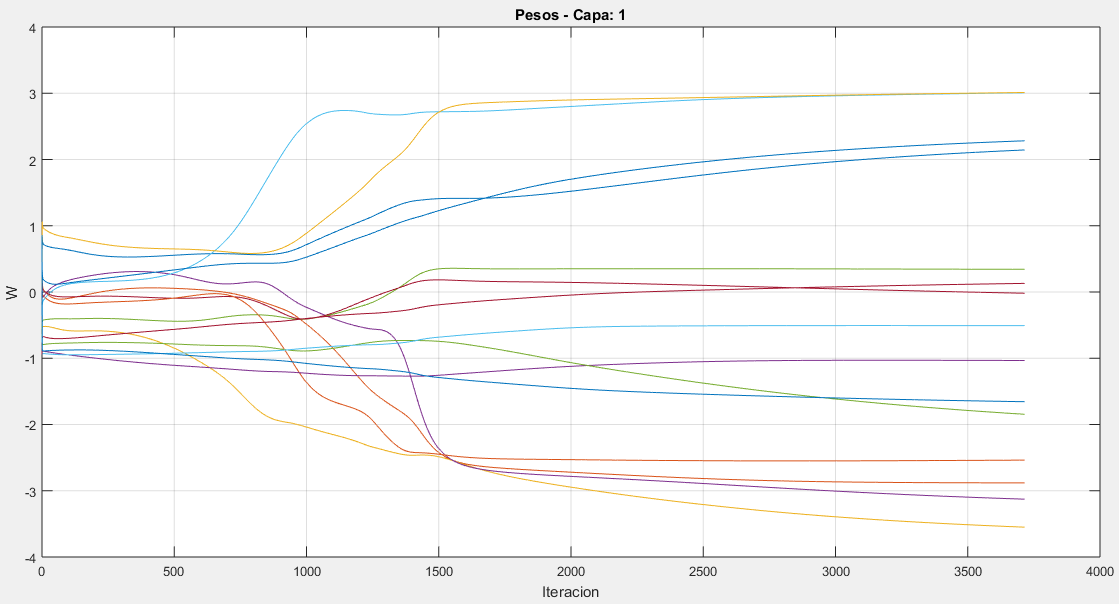


Figura . Pesos de la capa 1

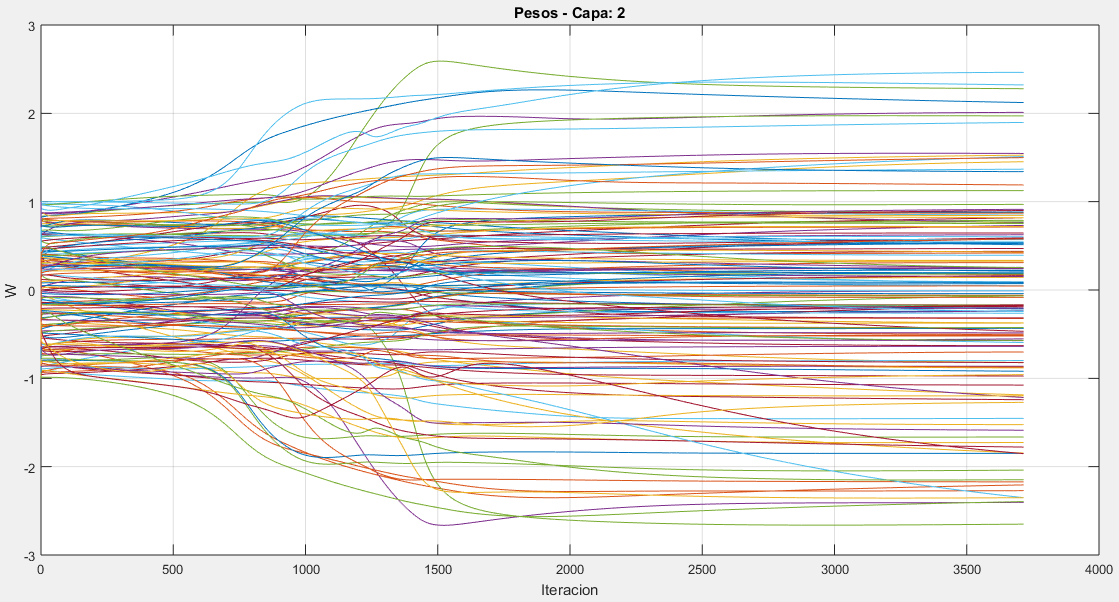


Figura . Pesos de la capa 2

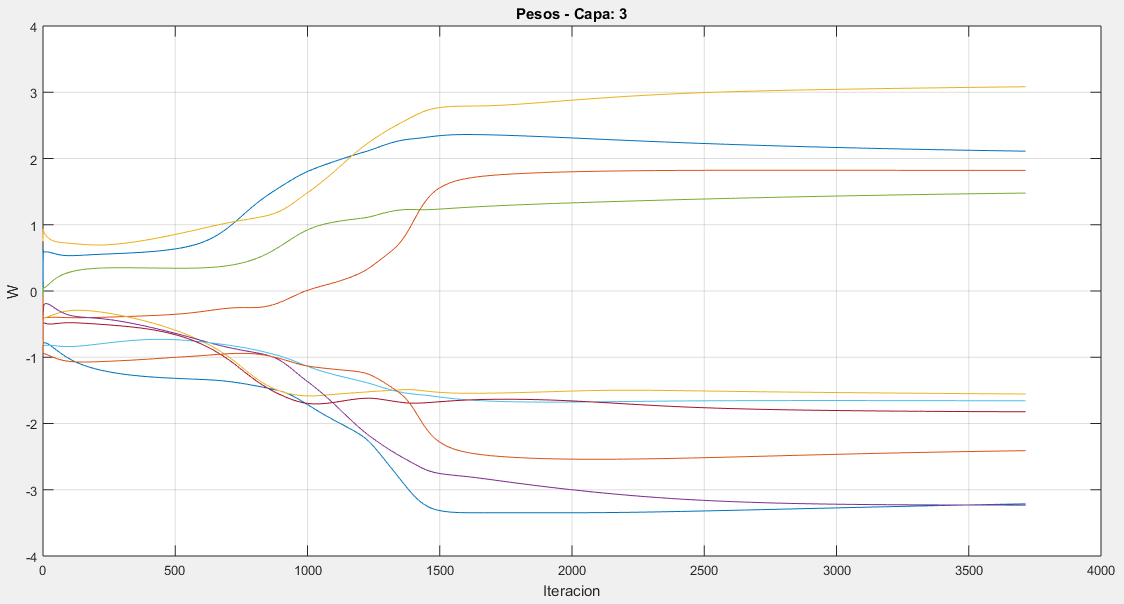


Figura . Pesos de la capa 3

Se muestra en las Figuras 15 y 16 la evolución de los bias por capa.

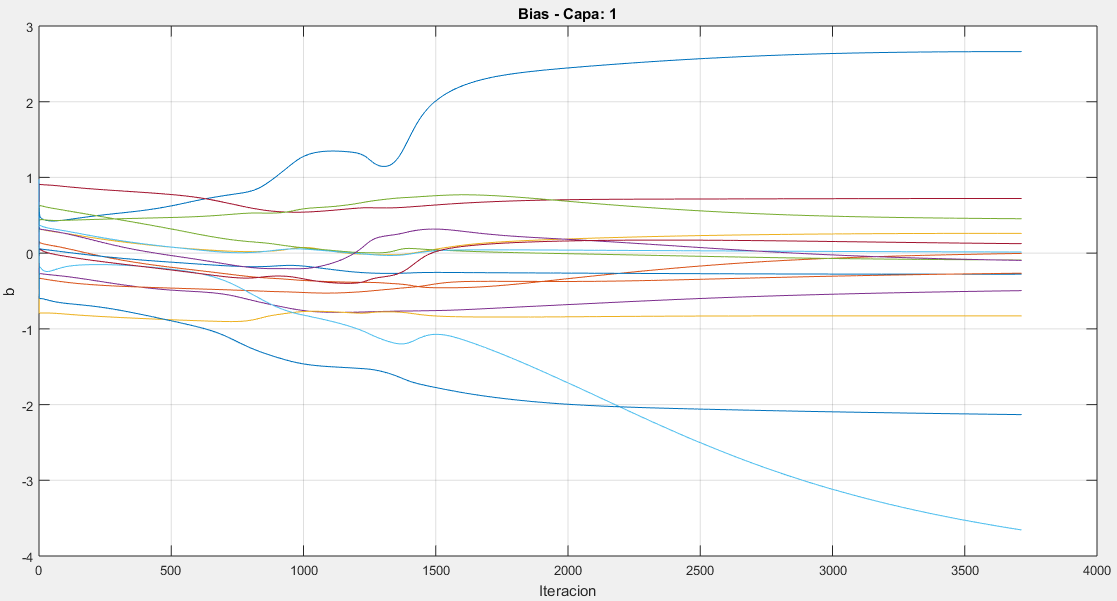


Figura . Bias de la capa 1

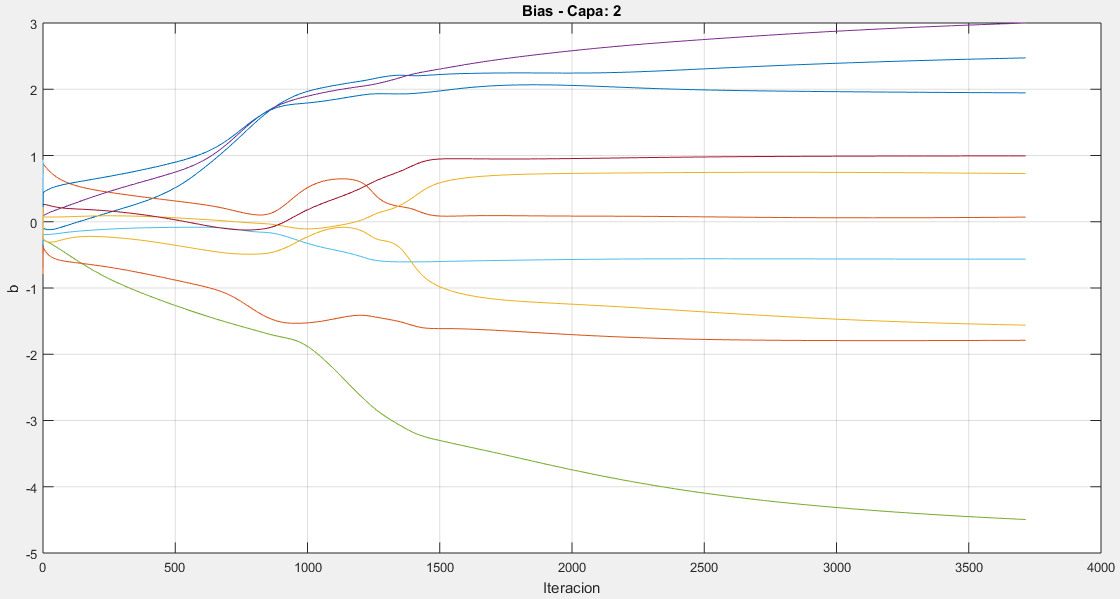


Figura . Bias de la capa 2

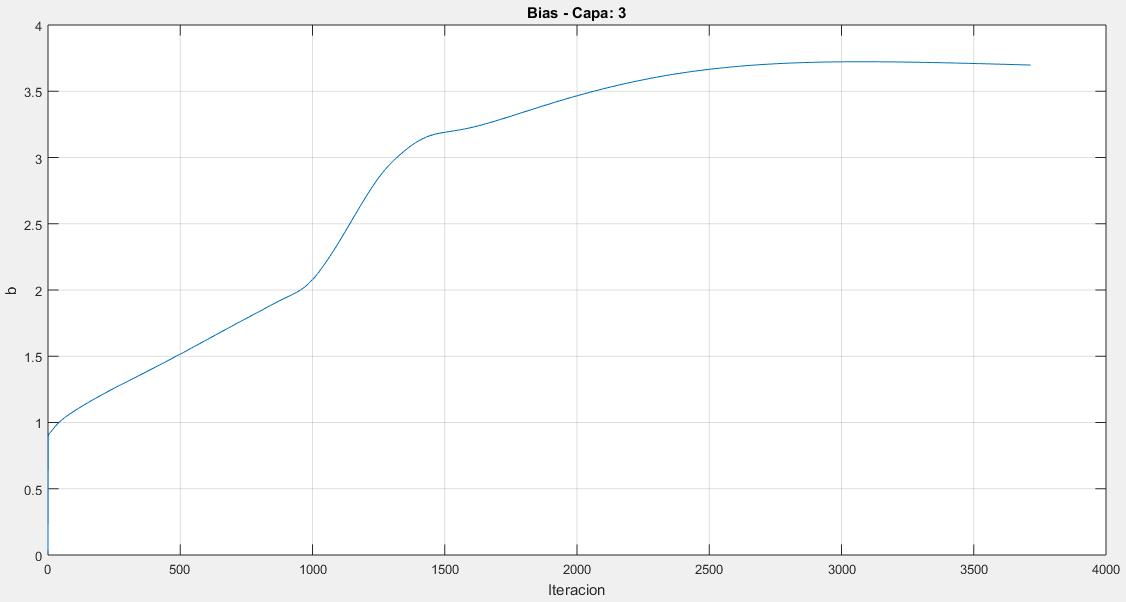


Figura . Bias de la capa 3

* **02\_POLINOMIO**

-0.01292

-0.01451

-0.01628

-0.01825

-0.02043

-0.02284

-0.02550

-0.02844

-0.03167

-0.03523

-0.03914

-0.04342

-0.04812

-0.05325

-0.05886

-0.06497

-0.07162

-0.07886

-0.08671

-0.09522

-0.10442

-0.11437

-0.12510

-0.13666

-0.14909

-0.16243

-0.17674

-0.19204

-0.20839

-0.22583

-0.24440

-0.26413

-0.28507

-0.30725

-0.33070

-0.35544

-0.38150

-0.40891

-0.43767

-0.46778

-0.49926

-0.53210

-0.56628

-0.60178

-0.63858

-0.67663

-0.71589

-0.75630

-0.79779

-0.84028

-0.88368

-0.92790

-0.97281

-1.01829

-1.06421

-1.11042

-1.15676

-1.20305

-1.24912

-1.29478

-1.33982

-1.38403

-1.42719

-1.46909

-1.50947

-1.54811

-1.58477

-1.61920

-1.65116

-1.68040

-1.70668

-1.72976

-1.74941

-1.76541

-1.77754

-1.78559

-1.78938

-1.78873

-1.78348

-1.77349

-1.75865

-1.73885

-1.71402

-1.68412

-1.64912

-1.60902

-1.56387

-1.51372

-1.45866

-1.39883

-1.33437

-1.26546

-1.19232

-1.11519

-1.03433

-0.95005

-0.86267

-0.77252

-0.67999

-0.58546

-0.48934

-0.39205

-0.29402

-0.19569

-0.09754

0.00000

0.09646

0.19138

0.28432

0.37483

0.46247

0.54683

0.62751

0.70412

0.77632

0.84375

0.90613

0.96317

1.01463

1.06030

1.10000

1.13359

1.16097

1.18207

1.19687

1.20536

1.20760

1.20367

1.19368

1.17779

1.15617

1.12905

1.09666

1.05928

1.01720

0.97075

0.92025

0.86605

0.80854

0.74809

0.68508

0.61990

0.55296

0.48465

0.41536

0.34547

0.27537

0.20543

0.13599

0.06741

0.00000

-0.06593

-0.13009

-0.19222

-0.25207

-0.30943

-0.36409

-0.41589

-0.46467

-0.51031

-0.55272

-0.59180

-0.62752

-0.65983

-0.68874

-0.71424

-0.73636

-0.75517

-0.77072

-0.78309

-0.79239

-0.79871

-0.80218

-0.80293

-0.80109

-0.79682

-0.79027

-0.78158

-0.77093

-0.75846

-0.74434

-0.72874

-0.71180

-0.69370

-0.67457

-0.65456

-0.63383

-0.61251

-0.59073

-0.56861

-0.54628

-0.52384

-0.50140

-0.47905

-0.45688

-0.43498

-0.41341

-0.39224

-0.37153

-0.35132

-0.33168

-0.31262

-0.29419

-0.27640

-0.25928

-0.24285

-0.22710

-0.21205

-0.19770

-0.18405

-0.17108

-0.15879

-0.14717

-0.13619

-0.12586

-0.11614

-0.10702

-0.09847

-0.09048

-0.08302

-0.07607

-0.06961

-0.06360

-0.05804

-0.05288

-0.04812

-0.04373

-0.03969

-0.03597

-0.03256

-0.02943

-0.02656

-0.02395

-0.02156

-0.01938

-0.01741

-0.01561

-0.01398

-0.01250

-0.01117

-0.00996

A continuación, se escriben los valores ingresados a la RNA:

* Entrada: 02\_polinomio\_entrada.txt
* Target: 02\_Polinomio\_Target.txt
* Rango: -4 4
* Arquitectura: 1 8 1
* Funciones: 2 1
* Alfa: 0.01
* Iteraciones máximas: 10000
* Valor mínimo del error: 0.000001
* Itval: 500
* Valor máximo de incrementos consecutivos: 5
* Distribución: 70 – 15 – 15

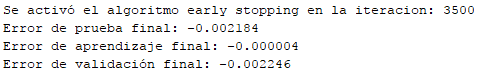


Figura . Algoritmo Early Stopping

Ahora, en las Figuras siguientes, se muestran los resultados gráficamente de la aproximación realizada por el MLP.

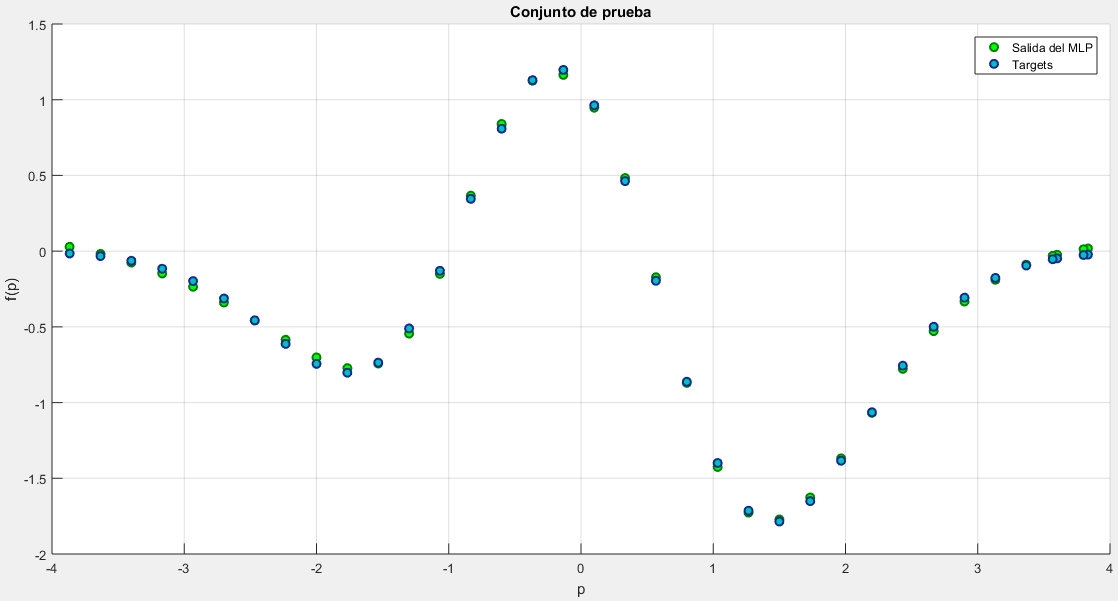


Figura . Subconjunto de prueba con sus targets

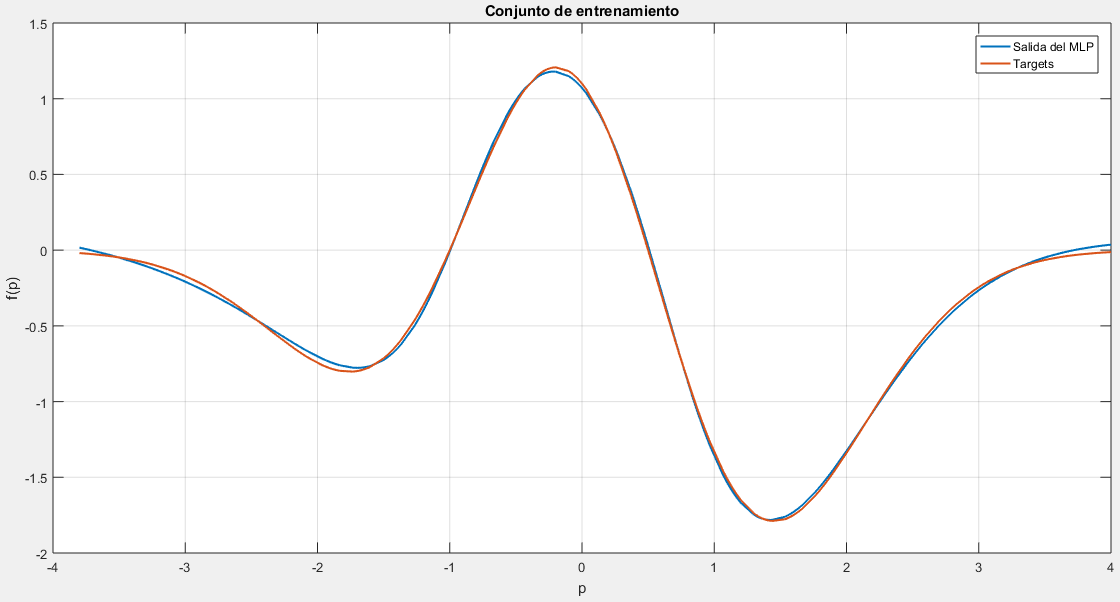


Figura . Subconjunto de entrenamiento con sus targets

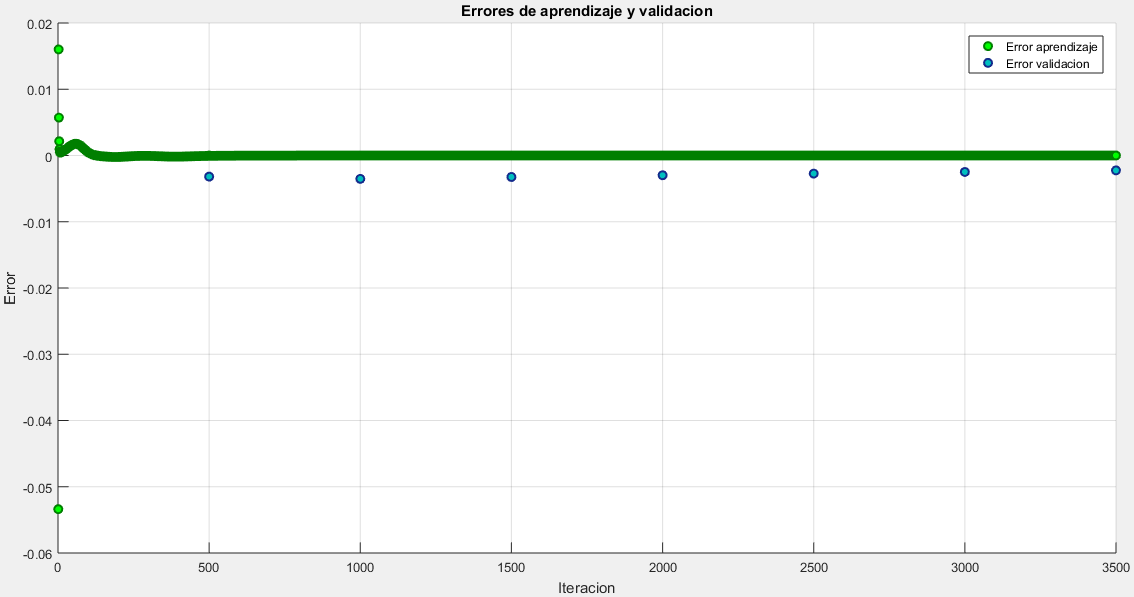


Figura . Errores de aprendizaje y de validación

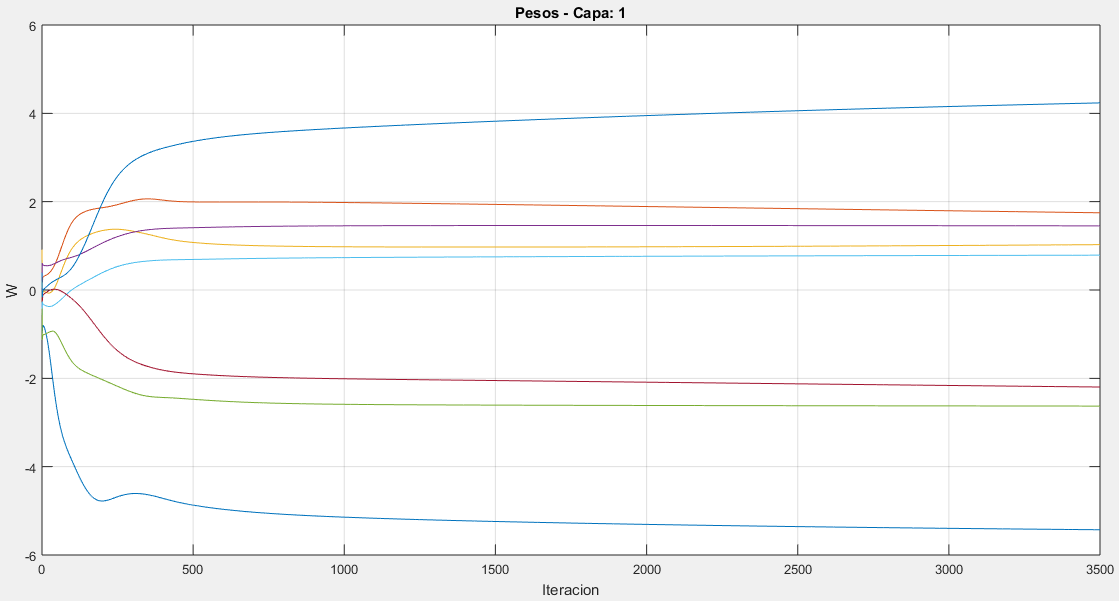


Figura . Pesos de la capa 1

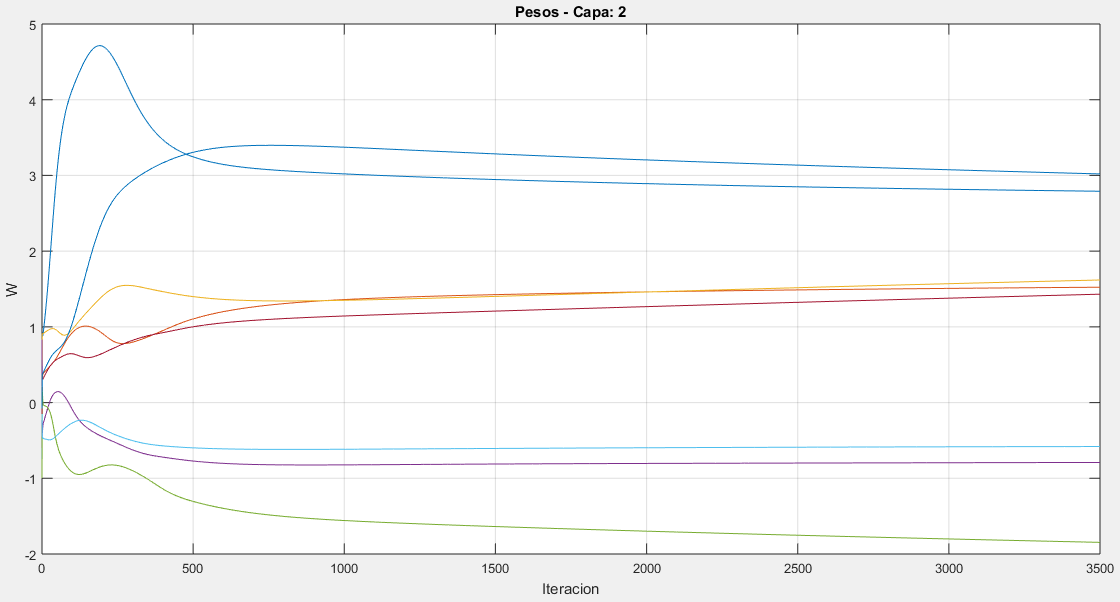


Figura . Pesos de la capa 2

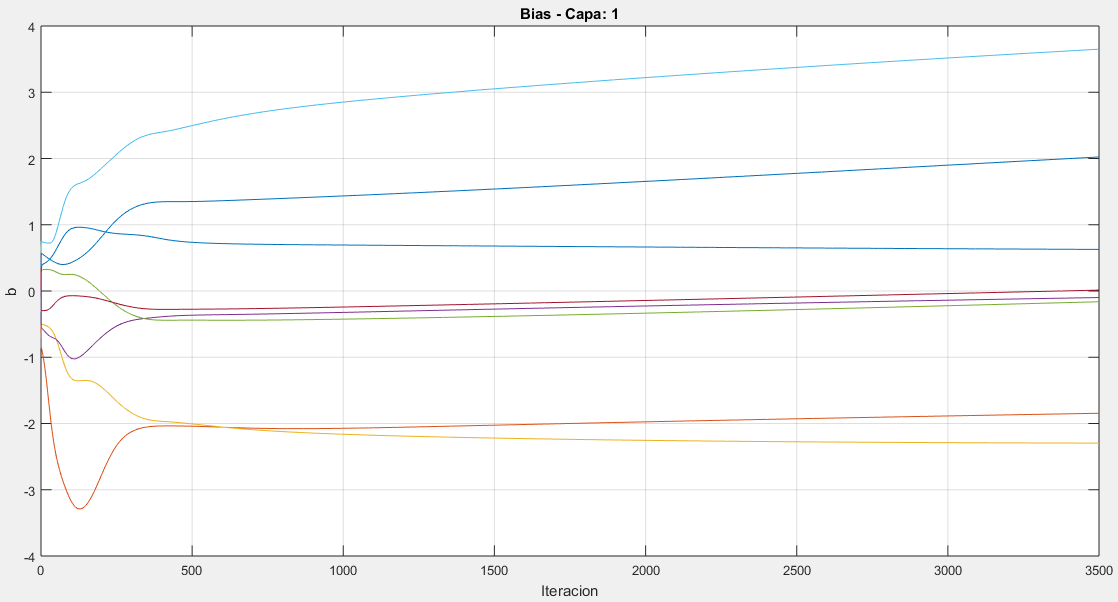


Figura . Bias de la capa 1

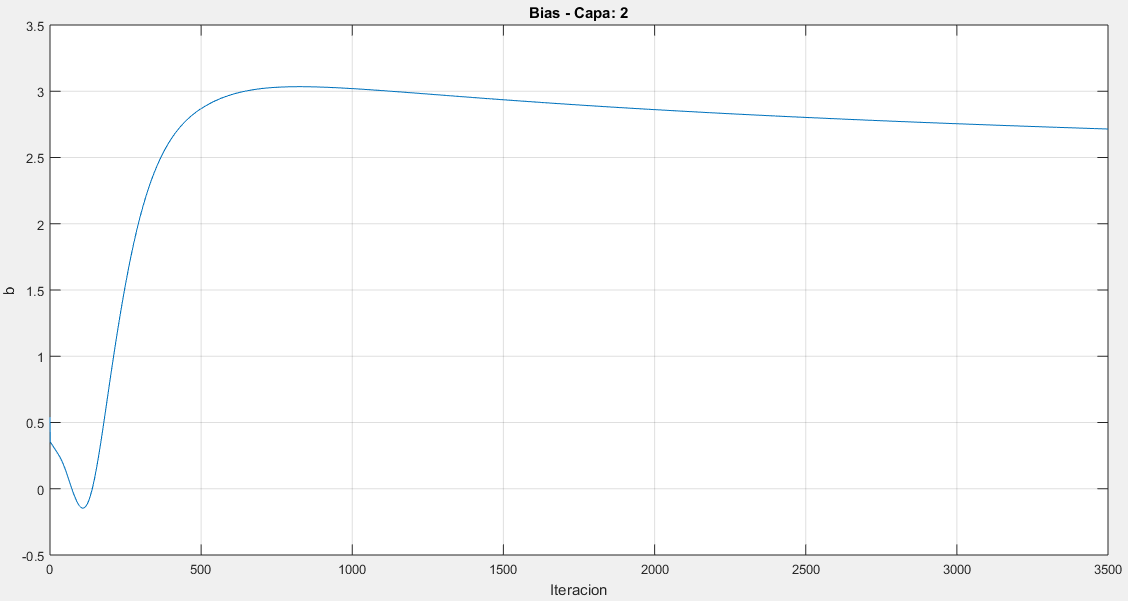


Figura . Bias de la capa 2

* **03\_POLINOMIO**

1

0.928632986

0.859165985

0.791568247

0.725809023

0.661858371

0.599686352

0.539262216

0.480557639

0.423544301

0.368191451

0.314472386

0.262357974

0.211821512

0.162833868

0.115369147

0.069400646

0.024900852

-0.018156939

-0.059797003

-0.100046044

-0.138929956

-0.176473016

-0.212701118

-0.247637729

-0.281307125

-0.313734391

-0.344942186

-0.374954786

-0.403794847

-0.431485838

-0.458049606

-0.483508809

-0.507885297

-0.531201726

-0.553479136

-0.574738566

-0.595001056

-0.614288454

-0.632620181

-0.650016467

-0.666498352

-0.682084447

-0.696794982

-0.710649379

-0.723665439

-0.735864202

-0.747261853

-0.757877812

-0.767729881

-0.776836673

-0.78521518

-0.792882396

-0.799855314

-0.806151737

-0.811788658

-0.816780643

-0.821145494

-0.824899396

-0.828056913

-0.83063423

-0.832646723

-0.834108956

-0.835037114

-0.835444953

-0.83534704

-0.834757939

-0.833691406

-0.8321612

-0.830181885

-0.827765599

-0.824926099

-0.821677142

-0.818030865

-0.814001024

-0.809598141

-0.80483597

-0.799725841

-0.794279893

-0.788510262

-0.782428278

-0.77604527

-0.769371758

-0.762419879

-0.755199344

-0.747721481

-0.739996003

-0.732033427

-0.723845083

-0.715439062

-0.706825886

-0.698015263

-0.689016905

-0.679839713

-0.670491778

-0.660983621

-0.651323333

-0.641519007

-0.631579544

-0.621513844

-0.611329192

-0.601033678

-0.590634587

-0.580140818

-0.569559655

-0.558897572

-0.548161851

-0.537360584

-0.526500246

-0.515587309

-0.504628247

-0.493630344

-0.482599263

-0.471541479

-0.460463464

-0.449370884

-0.438268593

-0.427163875

-0.416061584

-0.404966576

-0.393885324

-0.382822685

-0.371782703

-0.360771043

-0.34979337

-0.33885292

-0.327954549

-0.317103111

-0.306303463

-0.295558841

-0.284873291

-0.274251668

-0.263697208

-0.253213959

-0.242805157

-0.232474848

-0.222226268

-0.212062655

-0.201987245

-0.192004085

-0.182114792

-0.172322604

-0.162630757

-0.153042489

-0.143559417

-0.134183969

-0.124919383

-0.115768085

-0.106731694

-0.097812638

-0.089012535

-0.080333813

-0.071778899

-0.063348603

-0.055045352

-0.046870764

-0.038826459

-0.030913244

-0.02313274

-0.015486563

-0.007975523

-0.000601239

0.006634671

0.013732207

0.020690559

0.02750811

0.03418405

0.04071838

0.04710948

0.05335816

0.059461992

0.065422595

0.071237541

0.076908449

0.0824337

0.087813294

0.09304804

0.09813632

0.103078944

0.107876719

0.112528838

0.1170353

0.121396105

0.125612872

0.12968479

0.133612671

0.137396513

0.141037126

0.144535318

0.147890282

0.151104443

0.154177803

0.157110362

0.159903737

0.162557929

0.165075365

0.167454428

0.169698353

0.171807141

0.1737816

0.175622542

0.177332391

0.178910341

0.180359627

0.18167944

0.182872208

0.18393874

0.184880654

0.18569957

0.186395486

0.186970831

0.187427222

0.18776547

0.187987192

0.188093198

0.188086724

0.187966962

0.187737957

0.187399709

0.186953837

0.186402769

0.185747313

0.184989897

0.18413214

0.183174851

0.182120457

0.180970577

0.179726829

0.178391641

0.176965822

0.175451799

0.173851192

0.172165618

0.170397506

0.168547663

0.166619328

0.164613309

0.162531225

0.160376313

0.15815019

0.155853667

0.15348917

0.151058319

0.148564349

0.146008071

0.143391102

0.140715871

0.137984805

0.135199522

0.13236164

0.129472779

0.126535365

0.123551827

0.120523782

0.11745285

0.114340648

0.111189604

0.108002146

0.104779892

0.101523652

0.098236662

0.09492054

0.091577715

0.088208995

0.084815999

0.081402773

0.077968507

0.074517249

0.071049806

0.067567797

0.064073651

0.060569794

0.057056226

0.053536994

0.050012098

0.046483964

0.042954213

0.03942527

0.035897946

0.032374668

0.028857055

0.025347533

0.021846104

0.018355194

0.014876422

0.011412216

0.007963385

0.004532356

0.00111913

-0.002273056

-0.005642586

-0.008989457

-0.012311243

-0.015606325

-0.018873085

-0.022110714

-0.025317593

-0.028491294

-0.031631818

-0.034736737

-0.037805242

-0.040835714

-0.043826536

-0.046777706

-0.049685988

-0.052552192

-0.055373889

-0.058149462

-0.06087891

-0.063559805

-0.066192149

-0.068774322

-0.071305514

-0.073784109

-0.076209296

-0.078580266

-0.080895401

-0.083154701

-0.085356548

-0.087500131

-0.089585452

-0.091610083

-0.093574023

-0.095476464

-0.097316596

-0.09909361

-0.100807506

-0.102455856

-0.10403947

-0.105557538

-0.107008442

-0.108392992

-0.109709568

-0.110958171

-0.112137992

-0.11324903

-0.114290477

-0.115261523

-0.116162168

-0.116992412

-0.117751446

-0.11843927

-0.119055075

-0.11959967

-0.120071437

-0.120471184

-0.120798103

-0.121053002

-0.121235073

-0.121344316

-0.12138073

-0.121344316

-0.121235073

-0.121053002

-0.120798103

-0.120471184

-0.120071437

-0.11959967

-0.119055075

-0.11843927

-0.117751446

-0.116992412

-0.116162168

-0.115261523

-0.114290477

-0.11324903

-0.112137992

-0.110958171

-0.109709568

-0.108392992

-0.107008442

-0.105557538

-0.10403947

-0.102455856

-0.100807506

-0.09909361

-0.097316596

-0.095476464

-0.093574023

-0.091610083

-0.089585452

-0.087500131

-0.085356548

-0.083154701

-0.080895401

-0.078580266

-0.076209296

-0.073784109

-0.071305514

-0.068774322

-0.066192149

-0.063559805

-0.06087891

-0.058149462

-0.055373889

-0.052552192

-0.049685988

-0.046777706

-0.043826536

-0.040835714

-0.037805242

-0.034736737

-0.031631818

-0.028491294

-0.025317593

-0.022110714

-0.018873085

-0.015606325

-0.012311243

-0.008989457

-0.005642586

-0.002273056

0.00111913

0.004532356

0.007963385

0.011412216

0.014876422

0.018355194

0.021846104

0.025347533

0.028857055

0.032374668

0.035897946

0.03942527

0.042954213

0.046483964

0.050012098

0.053536994

0.057056226

0.060569794

0.064073651

0.067567797

0.071049806

0.074517249

0.077968507

0.081402773

0.084815999

0.088208995

0.091577715

0.09492054

0.098236662

0.101523652

0.104779892

0.108002146

0.111189604

0.114340648

0.11745285

0.120523782

0.123551827

0.126535365

0.129472779

0.13236164

0.135199522

0.137984805

0.140715871

0.143391102

0.146008071

0.148564349

0.151058319

0.15348917

0.155853667

0.15815019

0.160376313

0.162531225

0.164613309

0.166619328

0.168547663

0.170397506

0.172165618

0.173851192

0.175451799

0.176965822

0.178391641

0.179726829

0.180970577

0.182120457

0.183174851

0.18413214

0.184989897

0.185747313

0.186402769

0.186953837

0.187399709

0.187737957

0.187966962

0.188086724

0.188093198

0.187987192

0.18776547

0.187427222

0.186970831

0.186395486

0.18569957

0.184880654

0.18393874

0.182872208

0.18167944

0.180359627

0.178910341

0.177332391

0.175622542

0.1737816

0.171807141

0.169698353

0.167454428

0.165075365

0.162557929

0.159903737

0.157110362

0.154177803

0.151104443

0.147890282

0.144535318

0.141037126

0.137396513

0.133612671

0.12968479

0.125612872

0.121396105

0.1170353

0.112528838

0.107876719

0.103078944

0.09813632

0.09304804

0.087813294

0.0824337

0.076908449

0.071237541

0.065422595

0.059461992

0.05335816

0.04710948

0.04071838

0.03418405

0.02750811

0.020690559

0.013732207

0.006634671

-0.000601239

-0.007975523

-0.015486563

-0.02313274

-0.030913244

-0.038826459

-0.046870764

-0.055045352

-0.063348603

-0.071778899

-0.080333813

-0.089012535

-0.097812638

-0.106731694

-0.115768085

-0.124919383

-0.134183969

-0.143559417

-0.153042489

-0.162630757

-0.172322604

-0.182114792

-0.192004085

-0.201987245

-0.212062655

-0.222226268

-0.232474848

-0.242805157

-0.253213959

-0.263697208

-0.274251668

-0.284873291

-0.295558841

-0.306303463

-0.317103111

-0.327954549

-0.33885292

-0.34979337

-0.360771043

-0.371782703

-0.382822685

-0.393885324

-0.404966576

-0.416061584

-0.427163875

-0.438268593

-0.449370884

-0.460463464

-0.471541479

-0.482599263

-0.493630344

-0.504628247

-0.515587309

-0.526500246

-0.537360584

-0.548161851

-0.558897572

-0.569559655

-0.580140818

-0.590634587

-0.601033678

-0.611329192

-0.621513844

-0.631579544

-0.641519007

-0.651323333

-0.660983621

-0.670491778

-0.679839713

-0.689016905

-0.698015263

-0.706825886

-0.715439062

-0.723845083

-0.732033427

-0.739996003

-0.747721481

-0.755199344

-0.762419879

-0.769371758

-0.77604527

-0.782428278

-0.788510262

-0.794279893

-0.799725841

-0.80483597

-0.809598141

-0.814001024

-0.818030865

-0.821677142

-0.824926099

-0.827765599

-0.830181885

-0.8321612

-0.833691406

-0.834757939

-0.83534704

-0.835444953

-0.835037114

-0.834108956

-0.832646723

-0.83063423

-0.828056913

-0.824899396

-0.821145494

-0.816780643

-0.811788658

-0.806151737

-0.799855314

-0.792882396

-0.78521518

-0.776836673

-0.767729881

-0.757877812

-0.747261853

-0.735864202

-0.723665439

-0.710649379

-0.696794982

-0.682084447

-0.666498352

-0.650016467

-0.632620181

-0.614288454

-0.595001056

-0.574738566

-0.553479136

-0.531201726

-0.507885297

-0.483508809

-0.458049606

-0.431485838

-0.403794847

-0.374954786

-0.344942186

-0.313734391

-0.281307125

-0.247637729

-0.212701118

-0.176473016

-0.138929956

-0.100046044

-0.059797003

-0.018156939

0.024900852

0.069400646

0.115369147

0.162833868

0.211821512

0.262357974

0.314472386

0.368191451

0.423544301

0.480557639

0.539262216

0.599686352

0.661858371

0.725809023

0.791568247

0.859165985

0.928632986

1

A continuación, se escriben los valores ingresados a la RNA:

* Entrada/Target: 03-Polinomio\_Entrada\_y\_Target
* Rango: -4 4
* Arquitectura: 1 13 10 1
* Funciones: 3 2 1
* Alfa: 0.01
* Iteraciones máximas: 10000
* Valor mínimo del error: 0.000001
* Itval: 500
* Valor máximo de incrementos consecutivos: 5
* Distribución: 80 – 10 – 10

Ahora, en las Figuras siguientes, se muestran los resultados gráficamente de la aproximación realizada por el MLP.

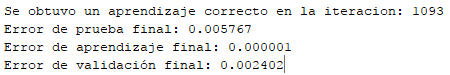


Figura . Mensaje de comprobación al usuario

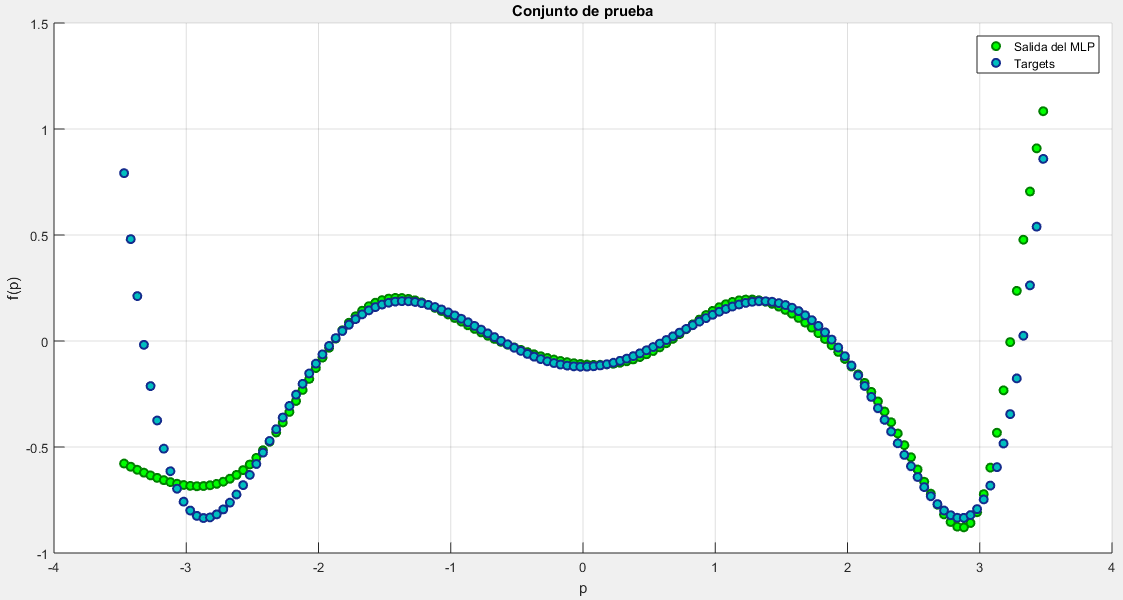


Figura . Subconjunto de prueba con sus targets

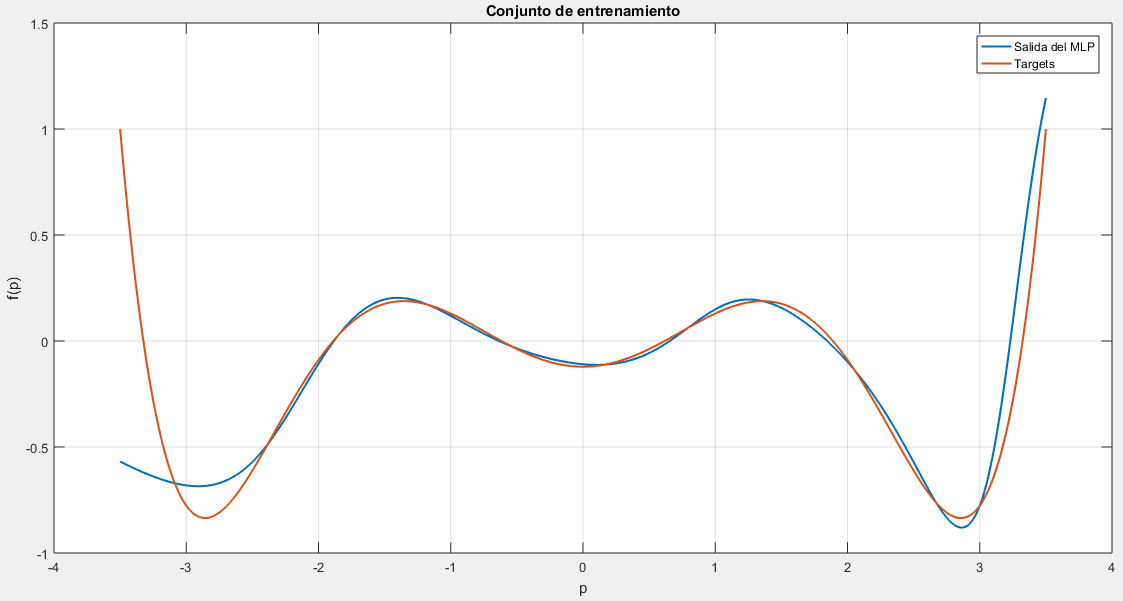


Figura . Subconjunto de entrenamiento con sus targets

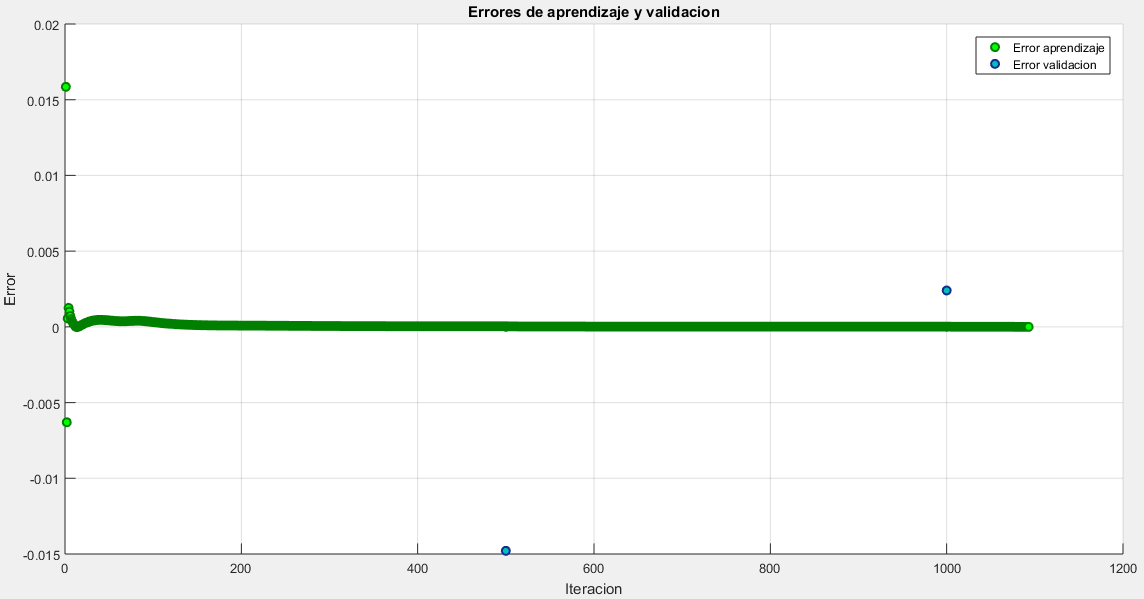


Figura . Errores de aprendizaje y de validación

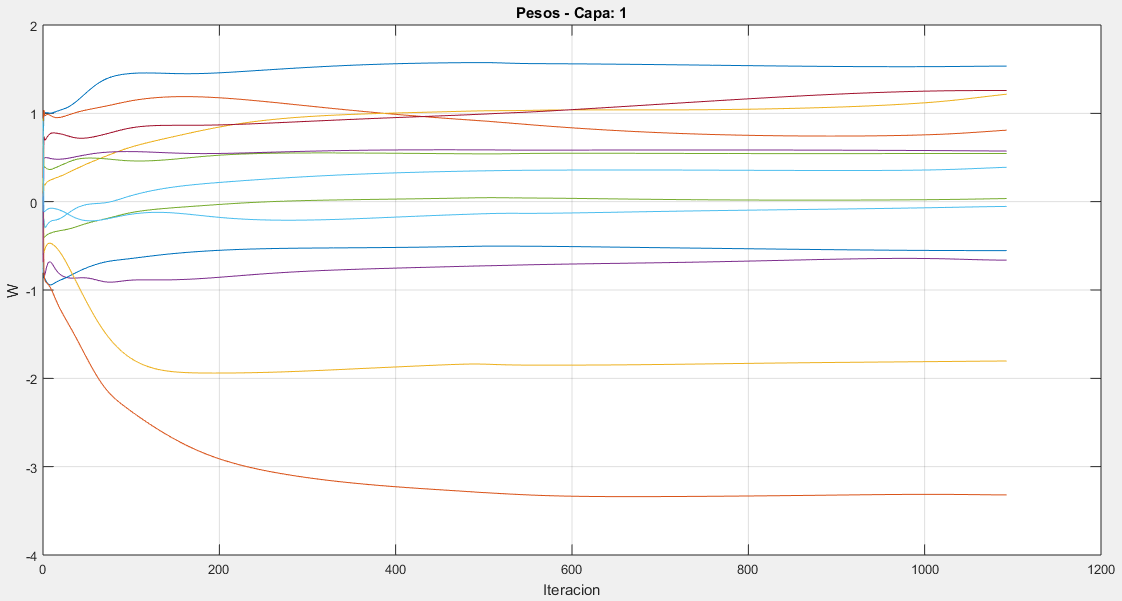


Figura . Pesos de la capa 1

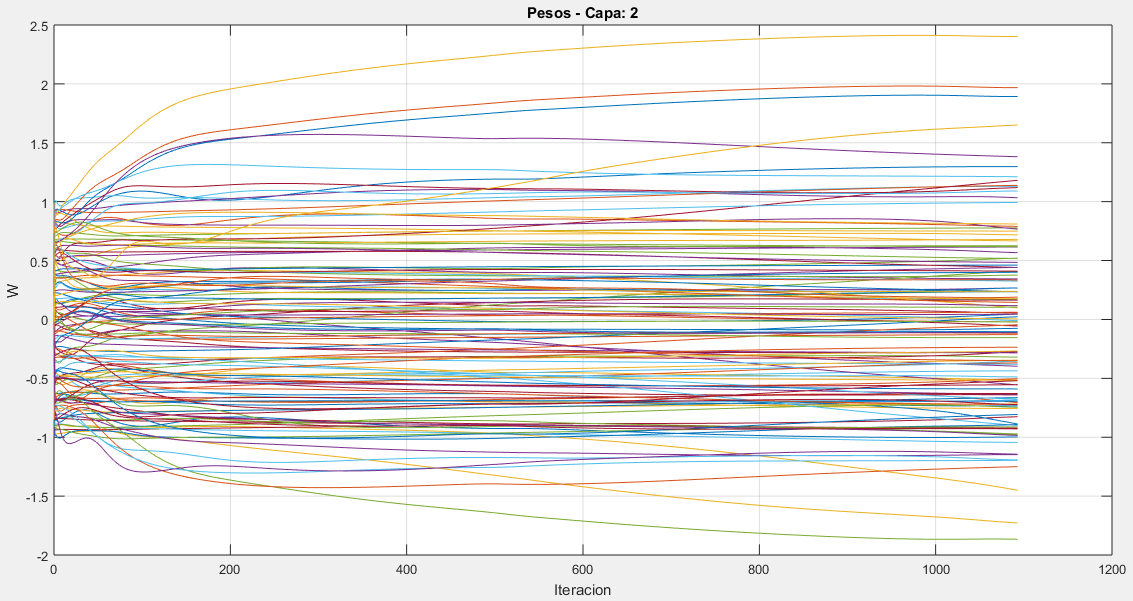


Figura . Pesos de la capa 2

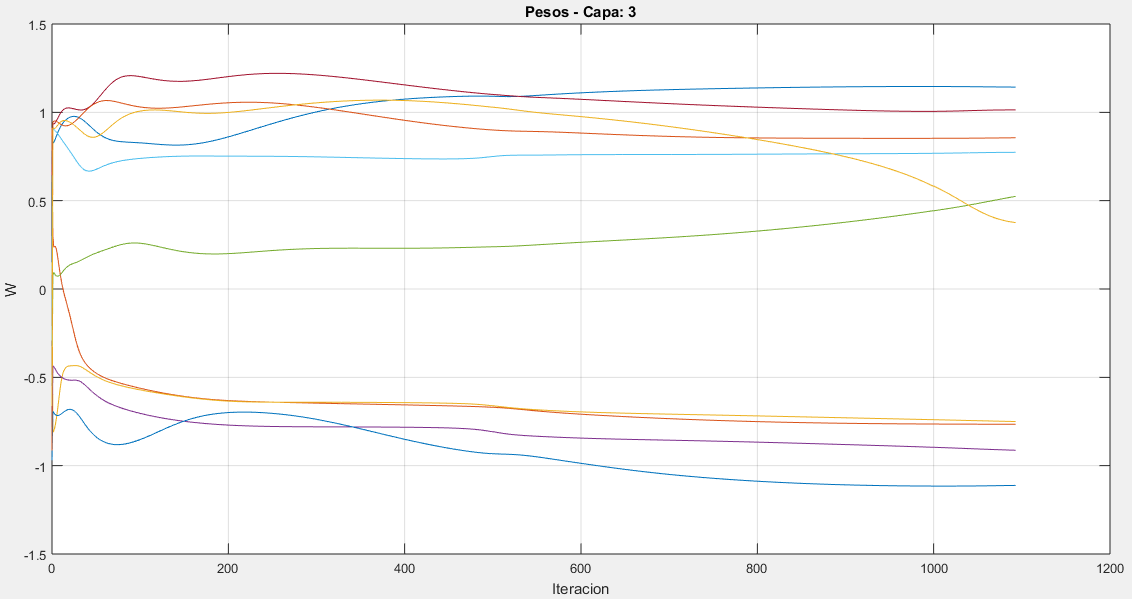


Figura . Pesos de la capa 3

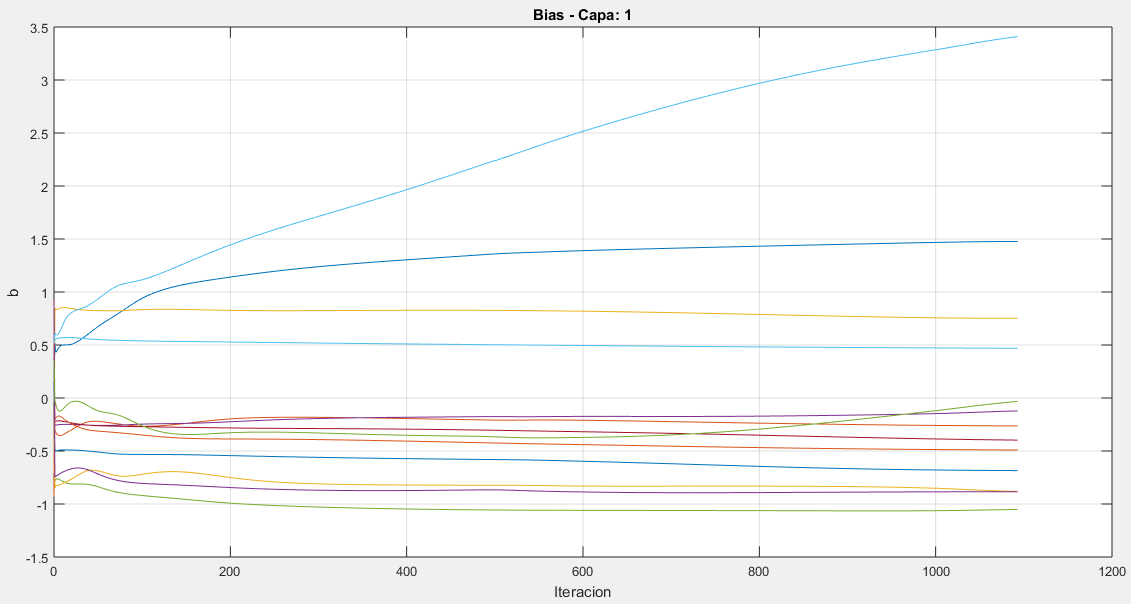


Figura . Bias de la capa 1

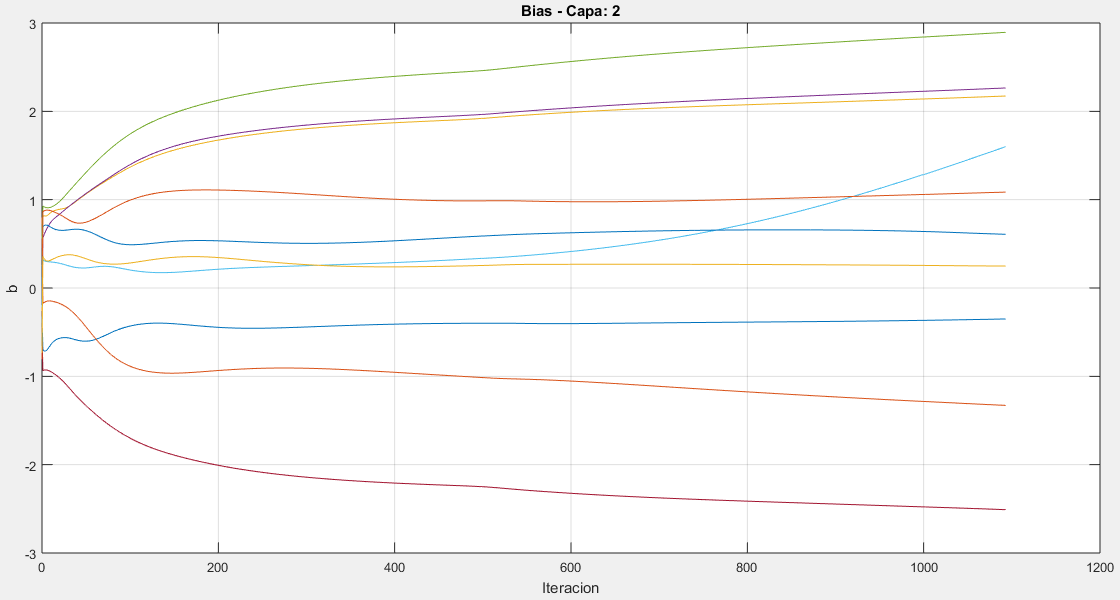


Figura . Bias de la capa 2

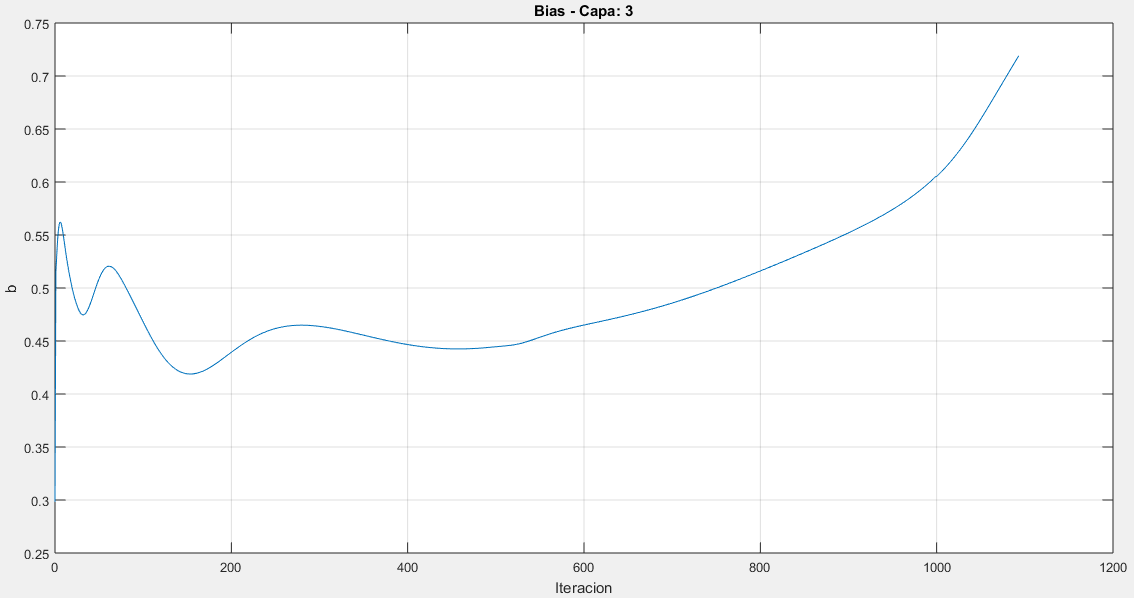


Figura . Bias de la capa 3

# Conclusiones:

Creo que lo más complicado personalmente, fue el estar haciendo las pruebas con distintas arquitecturas hasta encontrar alguna que aproximara bien cada una de las funciones propuestas, sin embargo, la programación de todo fue bastante complejo debido a la utilización de una estructura que yo no había utilizado, me refiero a “cell”, que en realidad son bastante útiles ya que pueden guardar cualquier tipo de dato, hasta otros cells y esto facilitó un poco el desarrollo de la práctica.

Creo que un perceptrón multicapa es una RNA muy útil ya que puede aproximar muchas señales como ya lo pudimos apreciar, sin embargo, creo que sería muchísimo más sencillo utilizar el Toolbox debido a que ya viene todo implementado, pero sin duda, el programarlo desde 0 nos hizo darnos cuenta de la complejidad que hay detrás de una RNA, además, de saber muy bien cómo es que está compuesta y cómo funciona su algoritmo de aprendizaje (que es muy distinto a todos los que habíamos visto), ya que el concepto de sensitividad fue difícil de implementar en esta práctica.

Lo más interesante del MLP es que al formarse de varias capas, le permite resolver problemas que no son linealmente separables, algo que personalmente me gustaría hacer con un MLP, sería el procesamiento de imágenes o encontrar patrones que a simple vista nosotros no podemos encontrar, ya que esto sería muy útil como por ejemplo detectar el patrón que tiene algún virus y saber cómo atacarlo, una nueva enfermedad o alguna cosa así.

En resumen, esta práctica fue interesante, pero sin duda un gran reto, debido a la complejidad de la programación que implicaba, ya que, se agregaron los conceptos nuevos de sensitividad y que, para calcularlas, hay que calcular también, la derivada de las funciones de activación que el usuario elija para crear una matriz que posteriormente se va a utilizar para el algoritmo de aprendizaje y actualizar los valores de pesos y bias para la siguiente iteración.

# Referencias:

**[1]** “Perceptrón Multicapa”, notas de clase de Neural Networks, Department of Engineering in Computer Systems, Escuela Superior de Cómputo, 2017.

**[2]** Martin T. Hagan, Howard B. Demuth, Mark H. Beale, Orlando De Jesús. *Neural Network Design*:Oklahoma, pp. 358 – 373.

**[3]** Math Works, “MATLAB”. Disponible en: <https://es.mathworks.com/products/matlab>.

**[4]** Edgardo Adrián Franco Martínez, “Software de Programación GNU”. Disponible en: <http://www.eafranco.com/?p=software/programacion/index.htm>.

# Código

**MLP.m**

%Limpieza de pantalla y variables

clc

clear

%Pedir al usuario el archivo de entrada (input.txt)

archivo = input ('Ingresa el archivo de entrada: ', 's');

p = importdata (archivo);

%Pedir al usuario el archivo de valores deseados (target.txt)

archivo = input ('Ingresa el archivo de los valores deseados: ', 's');

target = importdata (archivo);

%Pedir al usuario el rango de la se�al

rango = input ('Ingresa el rango de la se�al a aproximar: ', 's');

rango = str2num (rango);

%Pedir al usuario la arquitectura del MLP (M�ximo 3 capas ocultas)

arquitectura = input ('Ingresa la arquitectura separada por espacios: ', 's');

arquitectura = str2num (arquitectura);

%Pedir al usuario las funciones de activacion

fprintf ('Ingresa las funciones de activacion, donde:\n');

funciones\_activacion = input ('1. Purelin 2. Logsig 3. Tansig\n', 's');

funciones\_activacion = str2num (funciones\_activacion);

%Pedir al usuario el valor del factor de aprendizaje (alpha)

alpha = input ('Ingresa el valor del factor de aprendizaje (alpha): ');

%\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_CONDICIONES DE FINALIZACION\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

clc

itmax = input ('Ingresa el numero m�ximo de iteraciones (itmax): ');

Eit = input ('Ingresa el valor m�nimo del error por iteracion (Eit): ');

itval = input ('Ingresa cuantas iteraciones se realizar� una de validaci�n (itval): ');

numval = input ('Ingresa el valor m�ximo de incrementos consecutivos en el error de validaci�n (numval): ');

%\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_DIVISION EN 3 SUBCONJUNTOS\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

clc

fprintf ('Elija la distribuci�n de los datos.\n\n');

opcion = input ('1. 80 - 10 - 10\n2. 70 - 15 - 15\n\n');

numero\_datos = size (p);

numero\_datos = numero\_datos (1, 1);

valores = randperm (numero\_datos);

[entrenamiento, valores] = datos\_entrenamiento (opcion, valores, p, target);

[validacion, prueba] = datos\_validacion\_prueba (valores, p, target);

%Obtenemos el numero de elementos de cada subconjunto

numero\_datos\_entrenamiento = size (entrenamiento);

numero\_datos\_entrenamiento = numero\_datos\_entrenamiento (1, 1);

numero\_datos\_validacion = size (validacion);

numero\_datos\_validacion = numero\_datos\_validacion (1, 1);

numero\_datos\_prueba = size (prueba);

numero\_datos\_prueba = numero\_datos\_prueba (1, 1);

%Tama�o del vector de entrada p

R = arquitectura (1, 1);

%Calculamos el numero de capas que tendra el MLP

num\_capas = size (funciones\_activacion);

num\_capas = num\_capas (1, 2);

%Asignamos espacio a las matrices de pesos y bias

W = cell (num\_capas, 1);

b = cell (num\_capas, 1);

%\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ARCHIVOS PARA GRAFICACI�N\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

total\_archivos\_pesos = 0;

total\_archivos\_bias = 0;

**for** i = 1:num\_capas

**for** j = 1:(arquitectura (i+1))

**for** l = 1:(arquitectura (i))

total\_archivos\_pesos = (total\_archivos\_pesos + 1);

**end**

**end**

total\_archivos\_bias = (total\_archivos\_bias + 1);

**end**

archivos\_W = zeros (total\_archivos\_pesos, 1);

archivos\_b = zeros (total\_archivos\_bias, 1);

%Abrir archivos de pesos

archivo\_i = 1;

**for** i = 1:num\_capas

path = strcat (pwd, '/Capa ', num2str(i), '/Pesos/');

**if** ~exist (path, 'dir')

mkdir (path);

**end**

**for** j = 1:(arquitectura (i + 1))

**for** k = 1:(arquitectura (i))

archivo = strcat (path, '/Pesos', num2str (j), '\_', num2str (k), '.txt');

archivos\_W (archivo\_i) = fopen (archivo, 'w');

archivo\_i = (archivo\_i + 1);

**end**

**end**

**end**

%Abrir archivos de bias

archivo\_i = 1;

**for** i = 1:num\_capas

path = strcat (pwd, '/Capa ', num2str(i), '/Bias/');

**if** ~exist (path, 'dir')

mkdir (path);

**end**

**for** j = 1:(arquitectura (i + 1))

archivo = strcat (path, '/Bias', num2str (j), '.txt');

archivos\_b (archivo\_i) = fopen (archivo, 'w');

archivo\_i = (archivo\_i + 1);

**end**

**end**

%Asignar valores entre -1 y 1 a los pesos y bias

archivo\_Wi = 1;

archivo\_bi = 1;

**for** i = 1:num\_capas

W {i} = -1 + 2 \* rand (arquitectura (i + 1), arquitectura (i));

b {i} = -1 + 2 \* rand (arquitectura (i + 1), 1);

%Valores iniciales de pesos y bias en los archivos correspondientes

**for** j = 1:(arquitectura (i + 1))

**for** k = 1:(arquitectura (i))

fprintf (archivos\_W (archivo\_Wi), '%.4f\r\n', W {i} (j, k));

archivo\_Wi = (archivo\_Wi + 1);

**end**

**end**

**for** j = 1:(arquitectura (i + 1))

fprintf (archivos\_b (archivo\_bi), '%.4f\r\n', b {i} (j, 1));

archivo\_bi = (archivo\_bi + 1);

**end**

**end**

%Para guardar salidas, sensitividades y derivadas de cada capa

a = cell (num\_capas + 1, 1);

S = cell (num\_capas, 1);

F\_m = cell (num\_capas, 1);

%Se inicializan los errores de validacion

flag = 0;

error\_validacion\_anterior = 0;

error\_final\_aprendizaje = 0;

incrementos\_consecutivos = 0;

%Se inicializan matrices para errores

grafica\_error\_aprendizaje = zeros (itmax, 1);

grafica\_error\_validacion = zeros (ceil (itmax / itval), 1);

total\_iteraciones\_validacion = 0;

%\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ALGORITMO DE APRENDIZAJE\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**for** iteracion = 1:itmax

%Se resetea el valor del archivo W y b

archivo\_bi = 1;

archivo\_Wi = 1;

%Se resetea el valor del error de aprendizaje

error\_aprendizaje = 0;

%Si es iteracion de validacion, debe ser multiplo de itval

**if** mod (iteracion, itval) == 0

total\_iteraciones\_validacion = (total\_iteraciones\_validacion + 1);

error\_validacion\_actual = 0;

%Propagacion de los datos

**for** dato = 1:numero\_datos\_validacion

%Dato a propagar hacia adelante

a {1} = validacion (dato, 1);

%Propagacion hacia adelante del dato

**for** i = 1:num\_capas

a {i + 1} = funcion (W {i, 1}, a {i, 1}, b {i, 1}, funciones\_activacion (1, i));

**end**

%Se calcula el error de validacion para el dato i

error\_dato = (validacion (dato, 2) - a {num\_capas + 1, 1});

%error\_dato = abs (error\_dato);

%Se suma el error de validacion de cada dato

error\_validacion\_actual = (error\_validacion\_actual + error\_dato);

**end**

error\_validacion\_actual = (error\_validacion\_actual / numero\_datos\_validacion);

grafica\_error\_validacion (iteracion) = error\_validacion\_actual;

%Si ya hubo un incremento en el error de validacion

**if** (error\_validacion\_actual > error\_validacion\_anterior)

incrementos\_consecutivos = incrementos\_consecutivos + 1;

**if** incrementos\_consecutivos < numval

%Actualizacion del error anterior

error\_validacion\_anterior = error\_validacion\_actual;

error\_validacion\_actual = 0;

**else**

fprintf ('No se obtuvo un aprendizaje correcto de la red\n');

flag = 1;

fprintf ('\nEarly Stopping en la iteraci�n %d\n', iteracion);

**break**;

**end**

**else**

error\_validacion\_anterior = error\_validacion\_actual;

incrementos\_consecutivos = 0;

**end**

**else**

**for** dato = 1:numero\_datos\_entrenamiento

%Dato a propagar hacia adelante

a {1} = entrenamiento (dato, 1);

%Propagacion hacia adelante del dato

**for** i = 1:num\_capas

a {i + 1} = funcion (W {i, 1}, a {i, 1}, b {i, 1}, funciones\_activacion (1, i));

**end**

%Se calcula el error de validacion para el dato i

error\_dato = (entrenamiento (dato, 2) - a {num\_capas + 1, 1});

%error\_dato = abs (error\_dato);

%Se suma el error de validacion de cada dato

error\_aprendizaje = (error\_aprendizaje + error\_dato);

error\_aprendizaje = (error\_aprendizaje / numero\_datos\_entrenamiento);

%Calculo de sensitividades

F\_m {num\_capas} = matriz\_F (funciones\_activacion (1, num\_capas), arquitectura (1, num\_capas + 1), a {num\_capas + 1, 1});

S {num\_capas} = (-2 \* F\_m {num\_capas} \* error\_dato);

%Algoritmo Back Propagation

**for** i = (num\_capas - 1):-1:1

F\_m {i} = matriz\_F (funciones\_activacion (1, i), arquitectura (1, i + 1), a {i + 1, 1});

S {i} = F\_m {i, 1} \* (W {i+1, 1})' \* S {i + 1, 1};

**end**

%Actualizacion de pesos y bias

**for** i = num\_capas:-1:1

W {i, 1} = (W {i, 1} - (alpha \* S {i, 1} \* (a {i, 1})'));

b {i, 1} = (b {i, 1} - (alpha \* S {i, 1}));

**end**

**end**

error\_final\_aprendizaje = error\_aprendizaje;

grafica\_error\_aprendizaje (iteracion) = error\_aprendizaje;

**end**

%Imprimir valores de pesos y bias en archivo correspondiente

archivo\_Wi = 1;

archivo\_bi = 1;

**for** k = num\_capas:-1:1

**for** j = 1:(arquitectura (k + 1))

**for** l = 1:(arquitectura (k))

fprintf (archivos\_W (archivo\_Wi), '%.4f\r\n', W {k}(j, l));

archivo\_Wi = (archivo\_Wi + 1);

**end**

**end**

**for** j = 1:(arquitectura (k + 1))

fprintf (archivos\_b (archivo\_bi), '%.4f\r\n', b {k}(j, 1));

archivo\_bi = (archivo\_bi + 1);

**end**

**end**

%Condiciones de finalizaci�n por iteraci�n

**if** error\_aprendizaje < Eit && error\_aprendizaje > 0

fprintf ('Se obtuvo un aprendizaje exitoso en la iteracion: %d\n', iteracion);

**break**;

**end**

**end**

**if** flag == 1

archivo\_Wi = 1;

archivo\_bi = 1;

**for** k = num\_capas:-1:1

**for** j = 1:(arquitectura (k + 1))

**for** l = 1:(arquitectura (k))

fprintf (archivos\_W (archivo\_Wi), '%.4f\r\n', W {k}(j, l));

archivo\_Wi = (archivo\_Wi + 1);

**end**

**end**

**for** j = 1:(arquitectura (k + 1))

fprintf (archivos\_b (archivo\_bi), '%.4f\r\n', b {k}(j, 1));

archivo\_bi = (archivo\_bi + 1);

**end**

**end**

**end**

%\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_CERRAR ARCHIVOS DE PESOS Y BIAS\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**for** i = 1:total\_archivos\_pesos

fclose (archivos\_W (i));

**end**

**for** i = 1:total\_archivos\_bias

fclose (archivos\_b (i));

**end**

%\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_PROPAGACION CONJUNTO DE PRUEBA\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

error\_prueba = 0;

salida = ones (numero\_datos\_prueba, 1);

**for** i = 1:numero\_datos\_prueba

a {1} = prueba (i, 1);

**for** k = 1:num\_capas

a {k + 1} = funcion (W {k, 1}, a {k, 1}, b {k, 1}, funciones\_activacion (1, k));

**end**

aux = (prueba (i, 2) - a {num\_capas + 1, 1});

%aux = abs (aux);

error\_prueba = error\_prueba + (aux / numero\_datos\_prueba);

salida (i) = a {num\_capas + 1, 1};

**end**

%Impresion final de valores de los errores

fprintf ('Error final de aprendizaje = %.4f\n', error\_final\_aprendizaje);

fprintf ('Error final de validacion = %.4f\n', error\_validacion\_anterior);

fprintf ('Error final de prueba = %.4f\n', error\_prueba);

%\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_GRAFICACI�N\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

%ERRORES DE APRENDIZAJE Y VALIDACI�N

Prueba\_Errores = figure ('Name', 'Errores de aprendizaje y validaci�n');

figure (Prueba\_Errores);

grid on;

rango = 1:1:iteracion;

rango\_aux = itval:itval:(itval \* total\_iteraciones\_validacion);

%Verde oscuro

contorno\_validacion = [0 0.4980 0];

%Verde claro

relleno\_validacion = [0 1 0];

scatter (rango\_aux, grafica\_error\_validacion (itval:itval:total\_iteraciones\_validacion\*itval,1), 'MarkerEdgeColor',contorno\_validacion, 'MarkerFaceColor', relleno\_validacion, 'LineWidth',1.5);

hold on;

%Azul oscuro

contorno\_aprendizaje = [0.0784 0.1686 0.5490];

%Azul claro

relleno\_aprendizaje = [0 0.7490 0.7490];

scatter (rango, grafica\_error\_aprendizaje (1:iteracion,1), 'MarkerEdgeColor',contorno\_aprendizaje, 'MarkerFaceColor', relleno\_aprendizaje, 'LineWidth',1.5);

title ('Errores de aprendizaje y validacion');

xlabel ('Iteracion');

ylabel ('Valor del error');

legend ('Error validacion', 'Error aprendizaje');

%CONJUNTO DE PRUEBA CON TARGET

Prueba\_Graph = figure ('Name', 'Conjunto de prueba');

figure (Prueba\_Graph);

grid on;

rango = prueba (:, 1);

%Verde oscuro

contorno\_salida = [0 0.4980 0];

%Verde claro

relleno\_salida = [0 1 0];

scatter (rango, salida, 'MarkerEdgeColor',contorno\_salida, 'MarkerFaceColor', relleno\_salida, 'LineWidth',1.5);

hold on;

%Azul oscuro

contorno\_target = [0.0784 0.1686 0.5490];

%Azul claro

relleno\_target = [0 0.7490 0.7490];

scatter (rango, prueba (:, 2), 'MarkerEdgeColor',contorno\_target, 'MarkerFaceColor', relleno\_target, 'LineWidth',1.5);

title ('Conjunto de prueba');

xlabel ('p');

ylabel ('f (p)');

legend ('Salida del MLP', 'Target');

%PESOS

Pesos\_Graph = figure ('Name', 'Evoluci�n de los pesos');

grid on;

**for** i = 1:num\_capas

figure (Pesos\_Graph);

path = strcat (pwd, '/Capa ', num2str(i), '/Pesos/');

**for** j = 1:(arquitectura (i + 1))

**for** k = 1:(arquitectura (i))

archivo = strcat (path, '/Pesos', num2str (j), '\_', num2str (k), '.txt');

simb = strcat('W(',num2str(j),',',num2str(k),')');

evolucion\_pesos = importdata(archivo);

plot(evolucion\_pesos','DisplayName',simb);

hold on;

**end**

**end**

titulo = strcat('Pesos - capa',{' '},num2str(i));

title(titulo);

ylabel('W');

xlabel('Iteracion');

hold off

**end**

%BIAS

rango = 0:1:iteracion;

Bias\_Graph = figure ('Name', 'Evoluci�n de los bias');

grid on

**for** i = 1:num\_capas

figure (Bias\_Graph);

path = strcat (pwd, '/Capa ', num2str(i), '/Bias/');

**for** j = 1:(arquitectura (i+1))

archivo\_bias = strcat (path, '/Bias', num2str (j), '.txt');

simb = strcat('b(',num2str(j),')');

evolucion\_bias = importdata (archivo\_bias); % Identificador para la grafica

plot(rango, evolucion\_bias','DisplayName', simb);

hold on

**end**

titulo = strcat('Bias - capa',{' '},num2str(i));

title(titulo);

ylabel('b');

xlabel('Iteracion');

hold off

**end**

%ENTRENAMIENTO

figure

rango = entrenamiento(:,1);

%Verde oscuro

contorno\_salida = [0 0.4980 0];

%Verde claro

relleno\_salida = [0 1 0];

plot(salida,'MarkerEdgeColor',contorno\_salida, 'MarkerFaceColor', relleno\_salida, 'LineWidth',1.5);

grid on

hold on

%Azul oscuro

contorno\_target = [0.0784 0.1686 0.5490];

%Azul claro

relleno\_target = [0 0.7490 0.7490];

plot(entrenamiento(:,2),'MarkerEdgeColor',contorno\_target, 'MarkerFaceColor', relleno\_target, 'LineWidth',1.5);

title('Conjunto de entrenamiento');

ylabel('f(p)');

xlabel('p');

legend('Salida del MLP','Targets');

hold off

**funcion.m**

**function** a = funcion (W, a, b, funcion)

**if** funcion == 1

%Purelin

a = purelin ((W \* a) + b);

**elseif** funcion == 2

%Logsig

a = logsig ((W \* a) + b);

**else**

a = tansig ((W \* a) + b);

**end**

**end**

**matriz\_F.m**

**function** F = matriz\_F (funcion, neuronas, a)

**if** funcion == 1

%Purelin

F = diag (ones (1, neuronas));

**elseif** funcion == 2

%Logsig

F = diag (logsig ('dn', a, a));

**else**

%Tansig

F = diag (tansig ('dn', a, a));

**end**

**end**

**datos\_entrenamiento.m**

**function** [entrenamiento, nuevos\_val] = datos\_entrenamiento (opcion, valores, p, target)

**if** opcion == 1

aux = 0.8;

**else**

aux = 0.7;

**end**

total\_datos = size (valores);

total\_datos = total\_datos (1, 2);

%Numero de datos para el conjunto de entrenamiento

datos\_ent = ceil (total\_datos \* aux);

entrenamiento = zeros (datos\_ent, 2);

%Se comienzan a asignar los valores al conjunto de entrenamiento

**for** i = 1:datos\_ent

%Valores de entrenamiento en columna 1

entrenamiento (i, 1) = p (valores (1, i), 1);

%Valores de target en la columna 2

entrenamiento (i, 2) = target (valores (1, i), 1);

**end**

%Omitimos los datos ya ingresados para evitar repeticiones

nuevos\_val = valores (i + 1:total\_datos);

**end**

**datos\_validacion\_prueba.m**

**function** [validacion, prueba] = datos\_validacion\_prueba (valores, p, target)

total\_datos = size (valores);

total\_datos = total\_datos (1, 2);

%Numero de datos para el conjunto de validacion

datos\_val = ceil (total\_datos / 2);

datos\_prueba = (total\_datos - datos\_val);

%Numero de datos para el conjunto de prueba

validacion = zeros (datos\_val, 2);

prueba = zeros (datos\_prueba, 2);

%Se comienzan a asignar los valores al conjunto de validacion

**for** i = 1:datos\_val

%Valores de validacion en columna 1

validacion (i, 1) = p (valores (1, i), 1);

%Valores de target en la columna 2

validacion (i, 2) = target (valores (1, i), 1);

**end**

%Omitimos los datos ya ingresados para evitar repeticiones

valores = valores (i + 1:total\_datos);

%Se comienzan a asignar los valores al conjunto de prueba

**for** i = 1:datos\_prueba

%Valores de prueba en columna 1

prueba (i, 1) = p (valores (1, i), 1);

%Valores de target en la columna 2

prueba (i, 2) = target (valores (1, i), 1);

**end**

**end**