Forma, Flecha

Descripción generada automáticamente

Universidad Autónoma Chapingo

Departamento de Irrigación

****

**Departamento de Mecánica Agrícola**

**Ingeniería Mecatrónica Agrícola**

**Informe de prácticas**

**Asignatura:**

**Inteligencia Artificial**

**Nombre del profesor:**

**Luis Arturo Soriano Avendaño**

**Alumno:**

**Cocotle Lara Jym Emmanuel**

**[1710451-3]**

**Fecha: 10-Sep-2019**

**GRADO: GRUPO:**

**7° 7**

**Fecha de entrega: 17/12/2021**

Chapingo, Texcoco Edo. México

Índice

[Introducción 3](#_Toc90739388)

[Desarrollo 3](#_Toc90739389)

[Práctica 1: Red Neuronal Perceptrón 3](#_Toc90739390)

[Objetivo 3](#_Toc90739391)

[Programa 3](#_Toc90739392)

[Resultados 5](#_Toc90739393)

[Práctica 2: Red Perceptrón 5](#_Toc90739394)

[Objetivo 5](#_Toc90739395)

[Programa 5](#_Toc90739396)

[Resultados 7](#_Toc90739397)

[Práctica 3: Red Adaline-Widrow Hoff 7](#_Toc90739398)

[Objetivo 7](#_Toc90739399)

[Programa 7](#_Toc90739400)

[Resultados 9](#_Toc90739401)

[Práctica 4: Red Adaline: Reconocimiento de números 9](#_Toc90739402)

[Objetivo 9](#_Toc90739403)

[Programa 9](#_Toc90739404)

[Resultados 11](#_Toc90739405)

[Práctica 5: Red Adaline: Clasificación de números manuscritos. 11](#_Toc90739406)

[Objetivo 11](#_Toc90739407)

[Programa 11](#_Toc90739408)

[Resultados 13](#_Toc90739409)

[Práctica 6: Red Neuronal Multicapa 13](#_Toc90739410)

[Objetivo 13](#_Toc90739411)

[Programa 13](#_Toc90739412)

[Resultados 16](#_Toc90739413)

[Práctica 7: Descenso por gradiente estocástico 17](#_Toc90739414)

[Objetivo 17](#_Toc90739415)

[Programa 17](#_Toc90739416)

[Resultados 19](#_Toc90739417)

[Práctica 8: Red Neuronal de Propagación hacia atrás 20](#_Toc90739418)

[Objetivo 20](#_Toc90739419)

[Programa 20](#_Toc90739420)

[Resultados 22](#_Toc90739421)

[Práctica 9: Red neuronal multicapa: Clasificación de números. 23](#_Toc90739422)

[Objetivo 23](#_Toc90739423)

[Programa 23](#_Toc90739424)

[Resultados 25](#_Toc90739425)

[Conclusión 25](#_Toc90739426)

[Bibliografía 25](#_Toc90739427)

# Introducción

A través del tiempo la inteligencia artificial ha ido evolucionando de manera progresiva y con ello sus aplicaciones en diferentes campos, ya que en el seno de la inteligencia artificial como ciencia y tecnología se han ido acumulando conocimientos sobre cómo emular las diversas capacidades del ser humano para exhibir comportamientos inteligentes y se han desarrollado sistemas cada vez más perfeccionados que reproducen parcialmente dichas capacidades (Díez, R. P., Gómez, A. G., & de Abajo Martínez, N. 2001).

La enciclopedia de la inteligencia artificial la define como un campo de la ciencia y la ingeniería que se ocupa de la comprensión, desde el punto de vista informático, de lo que se denomina comúnmente comportamiento inteligente. También se ocupa de la creación de artefactos que exhiben este comportamiento.

La inteligencia artificial se basa en el análisis de cómo los seres humanos resuelven o buscan soluciones a cada uno de los innumerables problemas con los que se encuentran continuamente.

Los sistemas desarrollados con técnicas de inteligencia artificial deben enfrentarse con problemas para los que no se conoce la secuencia exacta de acciones que deben realizarse para encontrar su solución este tipo de estrategias de resolución de problemas se conocen como algoritmos de búsqueda.

A través de las prácticas de este informe se busca comprender los conceptos de la inteligencia artificial con ayuda de la programación en Python, con el objetivo de reforzar el conocimiento teórico que se mostró en las clases.

# Desarrollo

## Práctica 1: Red Neuronal Perceptrón

### Objetivo

Entrenamiento de una red neuronal perceptrón para la obtención de una matriz de pesos sinápticos y un vector de polarización además del error con el fin de poder reconocer dígitos en un display de 7 segmentos.

### Programa

*# Universidad Autónoma Chapingo*

*# Departamento de Ingeniería Mecánica Agrícola*

*# Ingeniería Mecatrónica Agrícola*

*# Jym Emmanuel Cocotle Lara*

*# 7° 7*

*#Librería para manejo de vectores y arreglos*

**import** **numpy** **as** **np**

*# Función escalón*

**def** **hardlim**(n):

**if** n > **0**:

value = **1**

**else**:

value = **0**

**return** value

*# Entradas*

*# a,b,c,d,e,f,g*

P = [[**1**,**1**,**1**,**1**,**1**,**1**,**0**], *# 0*

[**0**,**1**,**1**,**0**,**0**,**0**,**0**], *# 1*

[**1**,**1**,**0**,**1**,**1**,**0**,**1**], *# 2*

[**1**,**1**,**1**,**1**,**0**,**0**,**1**], *# 3*

[**0**,**1**,**1**,**0**,**0**,**1**,**1**], *# 4*

[**1**,**0**,**1**,**1**,**0**,**1**,**1**], *# 5*

[**1**,**0**,**1**,**1**,**1**,**1**,**1**], *# 6*

[**1**,**1**,**1**,**0**,**0**,**0**,**0**], *# 7*

[**1**,**1**,**1**,**1**,**1**,**1**,**1**], *# 8*

[**1**,**1**,**1**,**1**,**0**,**1**,**1**]] *# 9*

*# Valores esperados*

t\_pares = [**1**,**0**,**1**,**0**,**1**,**0**,**1**,**0**,**1**,**0**]

t\_mayores\_5 = [**0**,**0**,**0**,**0**,**0**,**0**,**1**,**1**,**1**,**1**]

t\_numeros\_p = [**0**,**0**,**1**,**1**,**0**,**1**,**0**,**1**,**0**,**0**]

t\_impares = [**0**,**1**,**0**,**1**,**0**,**1**,**0**,**1**,**0**,**1**]

t = t\_pares

*# Error*

e = np.ones(**10**)

*# Matriz de pesos sinápticos*

W = **2**\*np.random.rand(**1**,**7**)-**1**

*# Vector de polarización*

b = **2**\*np.random.rand(**1**)-**1**

*# Establecemos un for que reccorra el número de épocas*

**for** epocas **in** range(**500**):

*# Establecemos un for que reccorra el número de patrones de prueba*

**for** q **in** range(**10**):

a = hardlim(np.dot(W,P[q])+b) *# Salida de la neurona perceptrón*

e[q] = t[q]-a *# Error*

W += np.dot(e[q],P[q]).T *# Transpuesta*

b += e[q]

*# Resultados*

**print**(f"W: {W}")

**print**(f"b: {b}")

**print**(f"e: {e}")

### Resultados

Texto

Descripción generada automáticamente

Ilustración 1.- Resultados del entrenamiento de la red neuronal perceptrón.

Después del entrenamiento con 500 épocas se pudo obtener la matriz de pesos sinápticos el vector de polarización y el error siendo este último una matriz de ceros siendo esta matriz el error deseado.

## Práctica 2: Red Perceptrón

### Objetivo

Diseño de una red neuronal perceptrón para la clasificación de libros.

### Programa

*# Universidad Autónoma Chapingo*

*# Departamento de Ingeniería Mecánica Agrícola*

*# Ingeniería Mecatrónica Agrícola*

*# Jym Emmanuel Cocotle Lara*

*# 7° 7*

*# Librerí­as*

*# Para vectores y matrices*

**import** **numpy** **as** **np**

*# Para graficación*

**import** **matplotlib.pyplot** **as** **plt**

*# Función de activación*

**def** **hardlim**(n):

**if** n[**0**][**0**] > **0**:

value1 = **1**

**else**:

value1 = **0**

**if** n[**1**][**0**] > **0**:

value2 = **1**

**else**:

value2 = **0**

**return** [[value1],[value2]]

*# Arreglo de entradas*

P = np.array([[**0.7**,**1.5**,**2.0**,**0.9**,**4.2**,**2.2**,**3.6**,**4.5**], *# peso*

[**3**,**5**,**9**,**11**,**0**,**1**,**7**,**6**]]) *#frecuencia*

*# Valores esperados*

t = np.array([[**0**,**0**,**0**,**0**,**1**,**1**,**1**,**1**],

[**0**,**0**,**1**,**1**,**0**,**0**,**1**,**1**]])

*# error*

e = np.array(np.ones((**2**,**8**)))

*# inicializamos de forma aleatoria*

W = **2**\*np.random.rand(**2**,**2**)-**1** *# matriz de pesos sinápticos*

b = **2**\*np.random.rand(**2**,**1**)-**1** *# vector de polarizaciÃ³n*

**for** epocas **in** range(**40**): *# número de épocas*

**for** q **in** range(**8**): *# número de patrones de prueba*

a = hardlim(np.dot(W,P[:,q].reshape(-**1**,**1**))+b) *# convierte a vector de 1 columna*

e[:,q] = (t[:,q].reshape(-**1**,**1**)-a).T *# error*

W += np.dot(e[:,q].reshape(-**1**,**1**),P[:,q].reshape(-**1**,**1**).T)

b += e[:,q].reshape(-**1**,**1**)

*# Graficación del resultado*

fig,ax = plt.subplots()

*# Ligeros y poco usados*

ax.scatter(P[**0**][**0**],P[**1**][**0**],marker='^')

ax.scatter(P[**0**][**1**],P[**1**][**1**],marker='^')

*# Ligeros y muy usados*

ax.scatter(P[**0**][**2**],P[**1**][**2**],marker='s')

ax.scatter(P[**0**][**3**],P[**1**][**3**],marker='s')

*# Pesados y poco usados*

ax.scatter(P[**0**][**4**],P[**1**][**4**],marker='o')

ax.scatter(P[**0**][**5**],P[**1**][**5**],marker='o')

*# Pesados y muy usados*

ax.scatter(P[**0**][**6**],P[**1**][**6**],marker='\*')

ax.scatter(P[**0**][**7**],P[**1**][**7**],marker='\*')

points = np.arange(**0**,**6**,**0.01**)

*# Primera neurona*

ax.plot(points, (-b[**0**][**0**]/W[**0**,**1**])-(W[**0**,**0**]/W[**0**,**1**])\*points)

ax.plot(points, (-b[**1**][**0**]/W[**1**,**1**])-(W[**1**,**0**]/W[**1**,**1**])\*points)

plt.show()

### Resultados

Gráfico, Gráfico de dispersión

Descripción generada automáticamente

Ilustración 2.- Resultados de la clasificación de libros por medio de una red neuronal perceptrón

Como podemos observar en la figura dos la red neuronal puede clasificar los libros de manera adecuada, sin embargo, puede llegar a tener algunos errores al momento de la clasificación ya que las fronteras de decisión están muy cerca de los datos.

## Práctica 3: Red Adaline-Widrow Hoff

### Objetivo

Diseño de una red neuronal Adaline para la clasificación de libros de acuerdo con su peso y a su uso.

### Programa

*# Universidad Autónoma Chapingo*

*# Departamento de Ingeniería Mecánica Agrícola*

*# Ingeniería Mecatrónica Agrícola*

*# Jym Emmanuel Cocotle Lara*

*# 7° 7*

*# Librerí­as*

*# Para vectores y matrices*

**import** **numpy** **as** **np**

*# Para graficación*

**import** **matplotlib.pyplot** **as** **plt**

*# Función hardlim*

**def** **hardlim**(n):

**if** n[**0**][**0**] > **0**:

value1 = **1**

**else**:

value1 = -**1**

**if** n[**1**][**0**] > **0**:

value2 = **1**

**else**:

value2 = -**1**

**return** [[value1],[value2]]

*# Entradas*

P = np.array([[**0.7**,**1.5**,**2.0**,**0.9**,**4.2**,**2.2**,**3.6**,**4.5**],

[**3**,**5**,**9**,**11**,**0**,**1**,**7**,**6**]])

*# Vector aumentado*

Z = np.vstack([P,[**1**,**1**,**1**,**1**,**1**,**1**,**1**,**1**]])

*# Valores esperados : cambia 0 a -1*

T = np.array([[-**1**,-**1**,-**1**,-**1**,**1**,**1**,**1**,**1**],

[-**1**,-**1**,**1**,**1**,-**1**,-**1**,**1**,**1**]])

*# Algotimo de Widrow-Hoff*

R = np.dot(Z,Z.T)/**8** *# Q=8*

H = np.dot(Z,T.T)/**8**

X = np.linalg.inv(R) @ H

W = X[:**2**,:**2**].T

b = X[**2**:].reshape(-**1**,**1**)

*# Verificar la solución con el algoritmo perceptrón*

e = np.array(np.ones((**2**,**8**)))

**for** q **in** range(**8**):

*# Conversión del vector a un vector de 1 columna*

a = hardlim(np.dot(W,P[:,q].reshape(-**1**,**1**))+b)

e[:,q] = (T[:,q].reshape(-**1**,**1**)-a).T

**print**(e)

*# Graficación*

fig,ax = plt.subplots()

*# Ligeros y poco usados*

ax.scatter(P[**0**][**0**],P[**1**][**0**],marker='^')

ax.scatter(P[**0**][**1**],P[**1**][**1**],marker='^')

*#Ligeros y muy usados*

ax.scatter(P[**0**][**2**],P[**1**][**2**],marker='s')

ax.scatter(P[**0**][**3**],P[**1**][**3**],marker='s')

*#Pesados y poco usados*

ax.scatter(P[**0**][**4**],P[**1**][**4**],marker='o')

ax.scatter(P[**0**][**5**],P[**1**][**5**],marker='o')

*#Pesados y muy usados*

ax.scatter(P[**0**][**6**],P[**1**][**6**],marker='\*')

ax.scatter(P[**0**][**7**],P[**1**][**7**],marker='\*')

points = np.arange(**0**,**6**,**0.01**)

*# Primera neurona*

ax.plot(points, (-b[**0**][**0**]/W[**0**,**1**])-(W[**0**,**0**]/W[**0**,**1**])\*points)

ax.plot(points, (-b[**1**][**0**]/W[**1**,**1**])-(W[**1**,**0**]/W[**1**,**1**])\*points)

ax.set\_xlim([**0**,**6**])

ax.set\_ylim([-**2**,**14**])

plt.show()

### Resultados

Gráfico, Gráfico de dispersión

Descripción generada automáticamente

Ilustración 3.- Resultado de la clasificación de libros.

Como podemos observar la clasificación en este programa fue más acertada ya que las fronteras de decisión son más claras y por lo tanto la clasificación es más acertada.

## Práctica 4: Red Adaline: Reconocimiento de números

### Objetivo

Entrenamiento de una red Adaline para reconocimiento de números en un display de 7 segmentos.

### Programa

*# Universidad Autónoma Chapingo*

*# Departamento de Ingeniería Mecánica Agrícola*

*# Ingeniería Mecatrónica Agrícola*

*# Jym Emmanuel Cocotle Lara*

*# 7° 7*

*# Librerí­as*

*# Para vectores y matrices*

**import** **numpy** **as** **np**

*# Para graficación*

**import** **matplotlib.pyplot** **as** **plt**

*# Función hardlim*

**def** **hardlim**(n):

**if** n[**0**][**0**] > **0**:

value1 = **1**

**else**:

value1 = -**1**

**if** n[**1**][**0**] > **0**:

value2 = **1**

**else**:

value2 = -**1**

**if** n[**2**][**0**] > **0**:

value3 = **1**

**else**:

value3 = -**1**

**return** [[value1],[value2],[value3]]

*# a,b,c,d,e,f,g*

P = np.array([[**1**,**1**,**1**,**1**,**1**,**1**,**0**], *#0*

[**0**,**1**,**1**,**0**,**0**,**0**,**0**], *#1*

[**1**,**1**,**0**,**1**,**1**,**0**,**1**], *#2*

[**1**,**1**,**1**,**1**,**0**,**0**,**1**], *#3*

[**0**,**1**,**1**,**0**,**0**,**1**,**1**], *#4*

[**1**,**0**,**1**,**1**,**0**,**1**,**1**], *#5*

[**1**,**0**,**1**,**1**,**1**,**1**,**1**], *#6*

[**1**,**1**,**1**,**0**,**0**,**0**,**0**], *#7*

[**1**,**1**,**1**,**1**,**1**,**1**,**1**], *#8*

[**1**,**1**,**1**,**1**,**0**,**1**,**1**]]) *#9*

*# Transpuesta de P*

P = P.transpose()

*# Vector aumentado*

Z = np.vstack([P,np.ones(**10**)])

*# Valores esperados*

t\_pares = [**1**, -**1**, **1**, -**1**, **1**, -**1**, **1**, -**1**, **1**, -**1**]

t\_mayores\_5 = [-**1**,-**1**,-**1**,-**1**,-**1**,-**1**,**1**,**1**,**1**,**1**]

t\_numeros\_p = [-**1**,-**1**,**1**,**1**,-**1**,**1**,-**1**,**1**,-**1**,-**1**]

t\_impares = [-**1**,**1**,-**1**,**1**,-**1**,**1**,-**1**,**1**,-**1**,**1**]

*# Valores esperados*

T = np.array([t\_pares,t\_mayores\_5,t\_numeros\_p])

*# Algoritmo Adaline*

*# 10 = número de valores esperados*

R = np.dot(Z,Z.T)/**10**

H = np.dot(Z,T.T)/**10**

X = np.linalg.inv(R) @ H

W = X[:**7**,:**3**].T

b = X[**7**,:**3**].reshape(-**1**,**1**)

**print**(f"W: {W}")

**print**(f"b: {b}")

*# Verificando la solución*

e = np.ones((**3**,**10**))

**for** q **in** range(**10**):

a = hardlim(np.dot(W,P[:,q].reshape(-**1**,**1**))+b)

e[:,q] = (T[:,q].reshape(-**1**,**1**)-a).T

**print**(f"Error: {e}")

### Resultados

Texto

Descripción generada automáticamente

Ilustración 4.- Resultados del entrenamiento de la práctica 4.

Como podemos observar el entrenamiento logré adquirir la matriz de pesos sinápticos y el vector de polarización para que de esta forma se puedan reconocer los números de manera adecuada, sin embargo, el entrenamiento no fue del todo satisfactorio ya que en la matriz de error se encuentra un valor no deseado por lo que probablemente al momento de emplear los valores de la matriz de pesos sinápticos y el vector de polarización este pueda no identificar adecuadamente los números.

## Práctica 5: Red Adaline: Clasificación de números manuscritos.

### Objetivo

Entrenamiento de una red neuronal Adaline para el reconocimiento de números manuscritos.

### Programa

*# Universidad Autónoma Chapingo*

*# Departamento de Ingeniería Mecánica Agrícola*

*# Ingeniería Mecatrónica Agrícola*

*# Jym Emmanuel Cocotle Lara*

*# 7° 7*

*# Librerí­as*

*# Para vectores y matrices*

**import** **numpy** **as** **np**

*# Para graficación*

**import** **matplotlib.pyplot** **as** **plt**

*# Ruta del archivo*

data\_path = "Data/NumerosManuscritos.csv"

*# Establecemos a P con los datos del archivo*

P = np.loadtxt(data\_path,delimiter=",")

*# Convertimos el archivo en un arreglo*

P = np.array(P)

Z = np.vstack([P,np.ones((**1**,**5000**))])

*# Valores esperados*

T = np.vstack([np.ones((**1**,**500**)),-np.ones((**9**,**500**))]) *# Agregar filas*

**for** i **in** range(**1**,**10**):

T = np.hstack([T,np.vstack([-np.ones((i,**500**)),np.ones((**1**,**500**)),-np.ones((**9**-i,**500**))])]) *# agregar columnas*

*# Red neuronal Adaline*

*# 5000 = número de datos*

R = np.dot(Z,Z.T)/**5000**

*# 5000 = número de datos*

H = np.dot(Z,T.T)/**5000**

*# Multiplicación de matrices pseudoinversa*

X = np.linalg.pinv(R) @ H

*# Matriz de pesos sinápticos*

W = X[**0**:**400**,:].T

*# Vector de polarización*

b = X[**400**,:].reshape(-**1**,**1**)

*# Visualizar qué tan bien lo hizo*

index = np.ones((**1**,**5000**))

neurona\_sal = np.ones((**1**,**5000**))

**for** q **in** range(**5000**):

a = np.dot(W,P[:,q]).reshape(-**1**,**1**)+b

*# Función de activación*

neurona\_sal[:,q] = np.amax(a)

posicion = np.where(a==neurona\_sal[:,q])

index[:,q] = posicion[**0**]

y = np.zeros((**1**,**500**))

**for** j **in** range(**1**,**10**):

y = np.hstack([y,j\*np.ones((**1**,**500**))])

numeros\_aciertos = np.sum(y==index)

porcentaje\_aciertos = (numeros\_aciertos/**5000**)\***100**

**print**(f"{porcentaje\_aciertos} % de aciertos")

**for** k **in** range(**10**):

indice = np.round((**4999**\*np.random.rand(**1**)+**1**),**0**)

numero\_reconocido = index[:,int(indice)]

**print**(f"Número reconocido: {numero\_reconocido}")

pixels = P[:,int(indice)].reshape(**20**,**20**).T

plt.imshow(pixels, cmap="gray")

plt.title("Número reconocido: "+str(int(numero\_reconocido)))

plt.show()

### Resultados

Imagen en blanco y negro

Descripción generada automáticamente con confianza bajaImagen en blanco y negro

Descripción generada automáticamente con confianza bajaImagen en blanco y negro

Descripción generada automáticamente con confianza bajaImagen en blanco y negro

Descripción generada automáticamente con confianza bajaGráfico

Descripción generada automáticamenteGráfico

Descripción generada automáticamente con confianza mediaImagen en blanco y negro

Descripción generada automáticamente con confianza bajaImagen que contiene Código QR

Descripción generada automáticamenteGráfico

Descripción generada automáticamenteImagen que contiene Gráfico

Descripción generada automáticamente

Como podemos observar después del entrenamiento el programa ejecutó algunos ejemplos para reconocimiento de números y como se puede apreciar en las imágenes este reconocimiento no fue del todo fiable, sin embargo, tiene cierto grado de asertividad.

## Práctica 6: Red Neuronal Multicapa

### Objetivo

Entrenamiento de una red neuronal multicapa para realizar la función XOR

### Programa

*# Universidad Autónoma Chapingo*

*# Departamento de Ingeniería Mecánica Agrícola*

*# Ingeniería Mecatrónica Agrícola*

*# Jym Emmanuel Cocotle Lara*

*# 7° 7*

*# Librerí­as*

*# Para vectores y matrices*

**import** **numpy** **as** **np**

*# Para graficación*

**import** **matplotlib.pyplot** **as** **plt**

*# Función tangente sigmoidal*

**class** **TanSig**:

**def** **\_\_call\_\_**(self, x):

**return** np.tanh(x)

**def** **deriv**(self,x,y):

**return** **1.0** - np.square(y)

*# tangente hiperbólica*

**def** **tanh**(x):

t = (np.exp(x) - np.exp(-x))/(np.exp(x)+np.exp(-x))

**return** t

*# Entradas*

P = np.array([[**0**,**0**,**1**,**1**],

[**0**,**1**,**0**,**1**]])

*# Valores esperados*

T = np.array([-**1**,**1**,**1**,-**1**]) *# cambiamos 0 por -1*

*# Número de entrada de datos*

Q = **4**

*# Número de neuronas*

n1 = **34**

*# Epsilon: rango de valores iniciales*

ep = **1** *# parámetro que afectará a W y b iniciales*

*# Matriz de pesos sinápticos 1*

W1 = ep\***2**\*np.random.rand(n1,**2**)-**1**

*# Vector de polarización 1*

b1 = ep\***2**\*np.random.rand(n1,**1**)-**1**

*# Matriz de pesos sinápticos 2*

W2 = ep\***2**\*np.random.rand(**1**,n1)-**1**

*# Vector de polarización 2*

b2 = ep\***2**\*np.random.rand(**1**,**1**)-**1**

total\_epocas = **11000**

a2 = np.array(np.zeros((**1**,Q)))

error\_cuadratico\_medio = np.array(np.zeros((**1**,total\_epocas)))

alpha = **0.001**

**for** epocas **in** range(total\_epocas):

sum\_error = **0**

**for** q **in** range(Q):

*# Progagación de la entrada a la salida*

a1 = tanh(np.dot(W1,P[:,q].reshape(-**1**,**1**))+b1)

a2[:,q] = tanh(np.dot(W2,a1)+b2)

*# Retropropagación de la sensibilidad*

e = T[q]-a2[:,q]

*# Sensibilidad 2*

s2 = -**2**\*(**1**-(a2[:,q]\*\***2**))\*e

*# Sensibilidad 1*

s1 = (np.diag(**1**-(a1\*\***2**))\*W2.T)\*s2

*# Actualización de pesos sinapticos (W) y vectores de polarización (b)*

W2 = W2 - alpha\*s2\*a1.T

b2 = b2 - alpha\*s2

W1 = W1 - alpha\*s1\*P[:,q].reshape(-**1**,**1**).T

b1 = b1 - alpha\*s1

*# error cuadrático medio*

sum\_error = e\*\***2** + sum\_error

error\_cuadratico\_medio[:,epocas] = sum\_error/Q

*# Error cuadrático medio*

**print**(f"EQM: {error\_cuadratico\_medio}")

a\_verificacion = np.array(np.zeros((**1**,Q)))

*# Verificamos el resultado*

**for** q **in** range(Q):

a\_verificacion[:,q] = tanh(np.dot(W2,tanh(np.dot(W1,P[:,q].reshape(-**1**,**1**))+b1))+b2)

**print**(f"Valores esperados: {T}")

**print**(f"Valores de NN: {a\_verificacion}")

*# Frontera de decisión*

*# Gráfica de contorno*

u = np.linspace(-**2**,**2**,**100**)

v = np.linspace(-**2**,**2**,**100**)

z = np.array(np.zeros((**100**,**100**)))

**for** i **in** range(**100**):

**for** j **in** range(**100**):

z[i,j] = tanh(np.dot(W2,(tanh(np.dot(W1,[[u[i]],[v[j]]])+b1)))+b2)

x = np.arange(**0**,total\_epocas,**1**)

fig,(ax1,ax2) = plt.subplots(**1**,**2**)

ax1.set\_title('Error cuadrático medio')

ax1.plot(x,error\_cuadratico\_medio.reshape(-**1**,**1**))

ax1.set(xlabel='#épocas',ylabel='Error')

ax2.set\_title('Compuerta lógica XOR')

ax2.contour(u, v, z.T, **5**, linewidths = np.arange(-**0.9**, **0**, **0.9**))

ax2.scatter(P[**0**][**0**],P[**1**][**0**], marker='o')

ax2.scatter(P[**0**][**1**],P[**1**][**1**], marker='o')

ax2.scatter(P[**0**][**2**],P[**1**][**2**], marker='o')

ax2.scatter(P[**0**][**3**],P[**1**][**3**], marker='o')

*# Límites de los ejes*

ax2.set\_xlim([-**0.5**,**1.5**])

ax2.set\_ylim([-**0.5**,**1.5**])

plt.show()

### Resultados

Gráfico

Descripción generada automáticamente

Ilustración 5.- Resultados gráficos de la separabilidad de datos.

Como podemos observar para esta práctica es necesario realizar una separabilidad no lineal, para poder realizar la separación de datos de manera adecuada

Texto

Descripción generada automáticamente

Ilustración 6.- Valores de la red neuronal

En la ilustración 6 se muestran los valores del error cuadrático medio los valores esperados y los valores de la red neuronal.

## Práctica 7: Descenso por gradiente estocástico

### Objetivo

Realizar el entrenamiento de un algoritmo de descenso estocástico por gradiente para la clasificación de los patrones de 100 números.

### Programa

*# Universidad Autónoma Chapingo*

*# Departamento de Ingeniería Mecánica Agrícola*

*# Ingeniería Mecatrónica Agrícola*

*# Jym Emmanuel Cocotle Lara*

*# 7° 7*

*# Librerí­as*

*# Para vectores y matrices*

**import** **numpy** **as** **np**

*# Para graficación*

**import** **matplotlib.pyplot** **as** **plt**

*# Función de activación tangente sigmoidal*

**def** **tanh**(x):

value = (np.exp(x)-np.exp(-x))/(np.exp(x)+np.exp(-x))

**return** value

*# Ruta del archivo*

data\_path = ("Data/Clasificacion100.csv")

P = np.loadtxt(data\_path,delimiter=",")

P = np.array(P) *# (2 filas, 200 columnas)*

Q = **200**

*# Valores esperados*

T = np.ones((**1**,**200**))

T[:,**100**:] = -**1**

*# Valores iniciales*

n = **20** *# Número de neuronas*

ep = **1** *# Factor de escalamiento*

*# Parámetros*

*# Matrices de pesos sinapticos y vectores de polarización*

W1 = ep\***2**\*np.random.rand(n,**2**)-ep

b1 = ep\***2**\*np.random.rand(n,**1**)-ep

W2 = ep\***2**\*np.random.rand(**1**,n)-ep

b2 = ep\***2**\*np.random.rand(**1**,**1**)-ep

alfa = **0.01**

numero\_epocas = **500**

error\_qua\_medio = np.array(np.zeros((**1**,numero\_epocas)))

a2 = np.array(np.zeros((**1**,Q)))

**for** epocas **in** range(numero\_epocas):

sum\_error = **0**

**for** q **in** range(Q):

q = np.random.randint(**0**,**200**)

*# Propagación hacia delante (desde la entrada hasta la salida)*

a1 = tanh(np.dot(W1,P[:,q].reshape(-**1**,**1**))+b1)

a2[:,q] = tanh(np.dot(W2,a1)+b2)

*# Cálculo de la sensibilidad*

e = T[:,q]-a2[:,q]

s2 = -**2**\*(**1**-a2[:,q]\*\***2**)\*e

s1 = (np.diag(**1**-a1\*\***2**)\*W2.T)\*s2

*# Actualizando los pesos sinapticos*

W2 = W2 -alfa\*s2\*a1.T

b2 = b2 -alfa\*s2

W1 = W1 -alfa\*s1\*P[:,q].reshape(-**1**,**1**).T

b1 = b1 -alfa\*s1

sum\_error = e\*\***2**+sum\_error

error\_qua\_medio[:,epocas] = sum\_error/Q

**print**(f"Error cuadrático medio: {error\_qua\_medio}")

*# Visualizamos el error cuadrático medio*

x = np.arange(**0**,numero\_epocas,**1**)

fig, ax = plt.subplots()

ax.set\_title("Error cuadrático medio")

ax.plot(x,error\_qua\_medio.reshape(-**1**,**1**))

ax.set(xlabel="# Epocas",ylabel = "MSE")

*# Verificamos el algoritmo de backpropagation*

a\_verificacion = np.array(np.zeros((**1**,Q)))

**for** q **in** range(Q):

a\_verificacion[:,q] = tanh(np.dot(W2,tanh(np.dot(W1,P[:,q].reshape(-**1**,**1**))+b1))+b2)

x2 = np.arange(**0**,**200**,**1**)

fig2,ax2 = plt.subplots()

ax2.scatter(x2[**100**:],a\_verificacion[:,**100**:].reshape(-**1**,**1**),color='red')

ax2.scatter(x2[:**100**],a\_verificacion[:,:**100**].reshape(-**1**,**1**),color='blue')

*# Gráficas de contorno*

u = np.linspace(-**15**,**15**,**50**)

v = np.linspace(-**15**,**15**,**50**)

z = np.array(np.zeros((**50**,**50**)))

**for** i **in** range(**50**):

**for** j **in** range(**50**):

z[i,j] = tanh(np.dot(W2,tanh(np.dot(W1,[[u[i]],[v[j]]])+b1))+b2)

fig3,ax3 = plt.subplots()

ax3.scatter(P[**0**][**100**:],P[**1**][**100**:],marker="x",color='red')

ax3.scatter(P[**0**][:**100**],P[**1**][:**100**],marker="\*",color='blue')

ax3.contour(u,v,z.T,linewidths=np.arange(-**0.9**,**0**,**0.9**))

plt.show()

### Resultados

Gráfico, Histograma

Descripción generada automáticamente

Ilustración 7.- Valor del error cuadrático medio a través de las épocas.

Gráfico, Gráfico de dispersión

Descripción generada automáticamente

Ilustración 8.- Clasificación de los números.

Gráfico, Gráfico de dispersión

Descripción generada automáticamente

Ilustración 9.- Grafica de contornos con base en los números.

La clasificación de los números se puede observar claramente en la ilustración número 9, sin embargo, la eficacia de la clasificación puede mejorar.

## Práctica 8: Red Neuronal de Propagación hacia atrás

### Objetivo

Entrenamiento de una red neuronal multicapa para la aproximación de un modelo de regresión para los precios de una acción.

### Programa

*# Universidad Autónoma Chapingo*

*# Departamento de Ingeniería Mecánica Agrícola*

*# Ingeniería Mecatrónica Agrícola*

*# Jym Emmanuel Cocotle Lara*

*# 7° 7*

*# Librerí­as*

*# Para vectores y matrices*

**import** **numpy** **as** **np**

*# Para graficación*

**import** **matplotlib.pyplot** **as** **plt**

*# Función de activación sigmoidal*

**def** **sigmod**(x):

value = **1**/(**1**+np.exp(-x))

**return** value

*# Ruta del archivo*

data\_path = ("data/RegresionBursatil.csv")

Data = np.loadtxt(data\_path,delimiter=",")

Data = np.array(Data) *# datos de entrada, datos deseados*

P = Data[**0**]

T = Data[**1**]

Q = **31**

*# Número de neuronas de la primera capa*

n1 = **30**

*# Número de neuronas de la segunda capa*

n2 = **40**

ep = **1**

*# Matrices de pesos sinápticos y vectores de polarización*

W1 = ep\***2**\*np.random.rand(n1,**1**)-ep

b1 = ep\***2**\*np.random.rand(n1,**1**)-ep

W2 = ep\***2**\*np.random.rand(n2,n1)-ep

b2 = ep\***2**\*np.random.rand(n2,**1**)-ep

W3 = ep\***2**\*np.random.rand(**1**,n2)-ep

b3 = ep\***2**\*np.random.rand(**1**,**1**)-ep

alfa = **0.001**

total\_epocas = **4000**

error\_qua\_medio = np.array(np.zeros((**1**,total\_epocas)))

**for** epocas **in** range(total\_epocas):

sum\_error = **0**

**for** q **in** range(Q):

*# Algoritmo de propagación hacia adelante*

a1 = sigmod(np.dot(W1,P[q].reshape(-**1**,**1**))+b1)

a2 = sigmod(np.dot(W2,a1)+b2)

a3 = np.dot(W3,a2)+b3

*# Retropropagacion hacia atras o de las sensibilidades*

e = T[q]-a3

s3 = -**2**\*e

s2 = (np.diag((**1**-a2)\*a2)\*W3.T)\*s3

s1 = np.diag((**1**-a1)\*a1)\*(W2.T**@s2**)

*# Actualizar los pesos sinápticos y las polarizaciones*

W3 = W3-alfa\*s3\*a2.T

b3 = b3-alfa\*s3

W2 = W2-alfa\*s2\*a1.T

b2 = b2-alfa\*s2

W1 = W1-alfa\*s1\*P[q].T

b1 = b1-alfa\*s1

sum\_error = e\*\***2**+sum\_error

error\_qua\_medio[:,epocas] = sum\_error/Q

*# Verificando la solución de la red neuronal Backpropagation*

p = np.arange(**0**,**6**,**0.01**)

a3 = np.array(np.zeros((**1**,np.shape(p)[**0**])))

**for** q **in** range(np.shape(p)[**0**]):

a1 = sigmod(np.dot(W1,p[q])+b1)

a2 = sigmod(np.dot(W2,a1)+b2)

a3[:,q] = np.dot(W3,a2)+b3

*# Gráfica*

fig, ax = plt.subplots()

ax.scatter(Data[**0**],Data[**1**])

ax.plot(p,a3.reshape(-**1**,**1**),c='red')

x = np.arange(**0**,total\_epocas,**1**)

fig2, ax2 = plt.subplots()

ax2.plot(x,error\_qua\_medio.reshape(-**1**,**1**))

plt.show()

### Resultados

Interfaz de usuario gráfica

Descripción generada automáticamente

Ilustración 10.- Error cuadrático medio.

Gráfico, Gráfico de líneas

Descripción generada automáticamente

Ilustración 11.- Aproximación grafica de los datos de los diferentes precios.

Como podemos observar en la gráfica la aproximación no es del todo perfecta, ya que al principio y al final de la gráfica se puede ver una diferencia notable, el entrenamiento puede ser mejorado variando los valores de alfa, el número de épocas y el número de neuronas.

## Práctica 9: Red neuronal multicapa: Clasificación de números.

### Objetivo

Clasificación de números por medio de una red neuronal multicapa

### Programa

*# Universidad Autónoma Chapingo*

*# Departamento de Ingeniería Mecánica Agrícola*

*# Ingeniería Mecatrónica Agrícola*

*# Jym Emmanuel Cocotle Lara*

*# 7° 7*

*# Librerí­as*

*# Para vectores y matrices*

**import** **numpy** **as** **np**

*# Para graficación*

**import** **matplotlib.pyplot** **as** **plt**

*# Función tangente hiperbólica*

**def** **tanh**(x):

value = (np.exp(x)-np.exp(-x))/(np.exp(x)+np.exp(-x))

**return** value

*# Ruta del archivo*

data\_path = "Data/NumerosManuscritos.csv"

P = np.loadtxt(data\_path,delimiter=",")

P = np.array(P)

Q = **500**

T = np.vstack([np.ones((**1**,**500**)),-np.ones((**9**,**500**))])

**for** i **in** range(**1**,**10**):

T = np.hstack([T,np.vstack([np.ones((i,**500**)),np.ones((**1**,**500**)),-np.ones((**9**-i,**500**))])])

n1 = **25**

ep = **0.1**

*# Matrices de pesos sinápticos y vectores de polarización*

W1 = ep\*(**2**\*np.random.rand(n1,**400**)-**1**)

b1 = ep\*(**2**\*np.random.rand(n1,**1**)-**1**)

W2 = ep\*(**2**\*np.random.rand(**10**,n1)-**1**)

b2 = ep\*(**2**\*np.random.rand(**10**,**1**)-**1**)

alpha = **0.01**

total\_epocas = **100**

error\_cuadratico\_medio = np.array(np.zeros((**1**,total\_epocas)))

**for** epocas **in** range(total\_epocas):

sum\_error = **0**

**for** q **in** range(Q):

*# Propagación hacia adelante de la entrada a la salida*

a1 = tanh(np.dot(W1,P[:,q].reshape(-**1**,**1**))+b1)

a2 = tanh(np.dot(W2,a1)+b2)

*# Error*

e = T[:,q].reshape(-**1**,**1**) - a2

*# Propagación hacia atrás de las sensibilidades*

s2 = -**2**\*np.diag(**1**-a2\*\***2**)\*e

s1 = np.diag(**1**-a1\*\***2**)\*(W2.T**@s2**)

*# Actualizamos los parámetros*

W2 = W2 - alpha\*s2\*a1.T

b2 = b2 - alpha\*s2

W1 = W1 - alpha\*s1\*P[:,q].reshape(-**1**,**1**).T

b1 = b1 - alpha\*s1

sum\_error = e.T\*e

error\_cuadratico\_medio[:,epocas] = sum(sum\_error.reshape(-**1**,**1**))/Q

*# Visualizar el error cuadrático medio*

x = np.arange(**0**,total\_epocas,**1**)

fig,ax = plt.subplots()

ax.plot(x,error\_cuadratico\_medio.reshape(-**1**,**1**))

plt.show()

index = np.ones((**1**,**5000**))

neurona\_sal = np.ones((**1**,**5000**))

**for** q **in** range(**5000**):

a1 = tanh(np.dot(W1,P[:,q].reshape(-**1**,**1**))+b1)

a = np.amax(np.dot(W2,a1)+b2)

neurona\_sal[:,q] = a

posicion = np.where(a==neurona\_sal[:,q])

index[:,q] = posicion[**0**]

*# Valores reales*

y = np.zeros((**1**,**500**))

**for** j **in** range(**1**,**10**):

y = np.hstack([y,j\*np.ones((**1**,**500**))])

numero\_aciertos = np.sum(y==index)

**print**(f"Total aciertos: {str(numero\_aciertos)}")

porcentaje\_aciertos = (numero\_aciertos/**5000**)\***100**

**print**(f"{str(porcentaje\_aciertos)} % aciertos")

**for** k **in** range(**10**):

indice = np.round((**4999**\*np.random.rand(**1**)+**1**),**0**)

**print**(indice)

numero\_reconocido = index[:,int(indice)]

**print**(f"Número reconocido: {numero\_reconocido}")

pixels = P[:,int(indice)].reshape(**20**,**20**).T

plt.imshow(pixels,cmap="gray")

plt.title("Número reconocido: "+str(int(numero\_reconocido)))

plt.show()

### Resultados

Con respecto a la práctica 9, no se pudo tener un correcto entrenamiento ya que al momento de realizar la verificación los números no fueron clasificados correctamente

# Conclusión

Con ayuda de la inteligencia artificial podemos entrenar diferentes redes neuronales para realizar clasificación de diferentes objetos o datos que se tengan y a partir de esta clasificación poder realizar una toma de decisiones acertada.

# Bibliografía

Díez, R. P., Gómez, A. G., & de Abajo Martínez, N. (2001). Introducción a la inteligencia artificial: sistemas expertos, redes neuronales artificiales y computación evolutiva. Universidad de oviedo.