



Practica 3
Neurona Adaline

Seminario de solución de problemas de
inteligencia artificial 2

Clave del curso: I7041

NRC: 124882

Calendario: 2023-A

Sección: D01

Martínez Sepúlveda Alan Jahir

216569127

INCO


08/03/2023

Introducción:

Para la realización de esta práctica se pide la simulación de una neurona adaline, en la cual se generan las 3 funciones de activación, como lo son la lineal, sigmoideal y logística, tomando como valores de entrada un archivo con las entradas pertinentes en x1, x2 y su salida deseada

Desarrollo:

Para el desarrollo de esta practica, y como punto principal se toman los valores que se quieren entrenar, teniéndolos estos en un archivo .csv de la siguiente manera:

```
PRACTICAS > PRACTICA_3 >  entradas.csv
1    1,1,-1
2    1,-1,-1
3    -1,1,-1
4    -1,-1,1
5    |
```

Se crea la neurona Adaline con los siguientes parámetros:

- La función `__init__` es el constructor de la clase, que se ejecuta automáticamente cuando se crea un objeto de la clase. Tiene tres parámetros: `dim`, `aprendizaje` y `activacion`.
- `self.n` asigna el valor de `dim` a la variable `n`, que se utiliza para almacenar la cantidad de entradas de la red neuronal.
- `self.aprendizaje` asigna el valor de `aprendizaje` a la variable `aprendizaje`, que se utiliza para almacenar la tasa de aprendizaje de la red neuronal.
- `self.w` asigna un vector de pesos aleatorios a la variable `w` utilizando la función `np.random.rand()`. La dimensión de este vector es `dim x 1`, que es el número de entradas de la red neuronal.
- `self.b` asigna un sesgo aleatorio a la variable `b`.
- `self.activacion` asigna la función de activación a la variable `activacion`.

```
class Adaline:
    def __init__(self, dim, aprendizaje, activacion):
        self.n = dim
        self.aprendizaje = aprendizaje
        self.w = -1 + 2 * np.random.rand(dim, 1) # x = min + (max - min)*rand()
        self.b = -1 + 2 * np.random.rand()
        self.activacion = activacion
```

La función clasificar toma una entrada x, calcula la suma ponderada de los valores de entrada, agrega el sesgo y aplica una función de activación (lineal, logística o sigmoide) para obtener el resultado de la clasificación.

```
def clasificar(self, x):
    y = np.dot(self.w.transpose(), x) + self.b
    if self.activacion == "lineal":
        return y
    elif self.activacion == "logistica":
        return 1 / (1 + np.exp(-y))
    elif self.activacion == "sigmoideal":
        return np.tanh(y)
```

La función derivada_activacion calcula la derivada de la función de activación seleccionada para un valor de entrada y. Esta derivada se utiliza en el proceso de retropropagación de errores durante el entrenamiento de la red neuronal para ajustar los pesos y sesgos.

```
def derivada_activacion(self, y):
    if self.activacion == "lineal":
        return 1
    elif self.activacion == "logistica":
        return y * (1 - y)
    elif self.activacion == "sigmoideal":
        return 1 - y ** 2
```

Para el algoritmo de entrenamiento se utilizó:

- X es una matriz de entrada con n muestras y m características.
- y es un vector de salida deseado con n valores binarios (+1 o -1).
- epocas es el número de veces que se recorre todo el conjunto de entrenamiento.
- filename_prefix es un prefijo para el nombre de los archivos de imagen que se van a guardar.

- precision es una medida de la precisión mínima requerida (en términos de porcentaje) para detener el entrenamiento antes de que se alcancen todas las épocas.

En el ciclo principal de la función, se itera sobre el número de épocas especificadas. En cada época, se compara la salida real de la red neuronal con la salida deseada (y) para cada muestra en X. Si la salida de la red neuronal no coincide con la salida deseada, se ajustan los pesos y el sesgo utilizando la Regla Delta.

El proceso de ajuste de pesos y sesgo se realiza en un bucle anidado sobre cada muestra en X. Para cada muestra, se calcula la salida de la red neuronal y se compara con la salida deseada (y). A continuación, se calcula el error y se utiliza la derivada de la función de activación para ajustar los pesos y el sesgo. Este proceso se repite hasta que la salida de la red neuronal coincide con la salida deseada para todas las muestras en X.

```
def train(self, X, y, epocas, filename_prefix, precision=None):
    n, m = X.shape
    for i in range(epocas):
        # convertir los valores continuos de salida en valores binarios
        # comparar los vectores de salida de la red neuronal y los valores de la etiqueta de salida
        while not np.array_equal(np.sign(self.clasificar(X)), np.sign(y)):
            for j in range(m):
                y_pred = self.clasificar(X[:, j])
                error = y[j] - y_pred
                delta = error * self.derivada_activacion(y_pred)
                self.w += self.aprendizaje * delta * X[:, j].reshape(-1, 1)
                self.b += self.aprendizaje * delta
            graph(self.w)
            if np.array_equal(np.sign(self.clasificar(X)), np.sign(y)):
                break
        # if precision is not None and np.mean(np.allclose(self.clasificar(X), y)) >= float(precision):
        if precision is not None and np.mean(np.sign(self.clasificar(X)) == np.sign(y)) >= float(precision):
            print(f"Precision del {precision*100}%!")
            return
        plt.savefig(filename_prefix + str(i) + '.png')
        # Salir del bucle si se ha logrado una clasificación correcta
        if np.sign(self.clasificar(X)) == np.sign(y).all():
            print("¡Todas las clasificaciones son correctas!")
            break
```

Después solo resta leer los archivos que tenemos en nuestro archivo .csv, llamar a nuestra neurona Adaline con los parámetros necesarios, en nuestro caso se pasan las entradas, se utiliza un coeficiente de aprendizaje de 0.1 y se utiliza la función de activación lineal.

Solo resta entrenar nuestra neurona pasándole como parámetros el número de época, en nuestro caso 10, el nombre que tomara el archivo de la grafica que guardara, y por último la precisión que queremos que nuestra neurona tenga, para el ejemplo yo le puse un 95% de efectividad.

```
# Obtener el numero de filas en la entrada del archivo
with open("entradas.csv") as file:
    rows = len(file.readlines())

# Obtener el numero de columnas en la entrada del archivo
with open("entradas.csv") as f:
    reader = csv.reader(f, delimiter=',')
    columns = len(next(reader))

adaline = Adaline(columns - 1, 0.1, "lineal") # activacion por defecto es logistica
arreglo = []

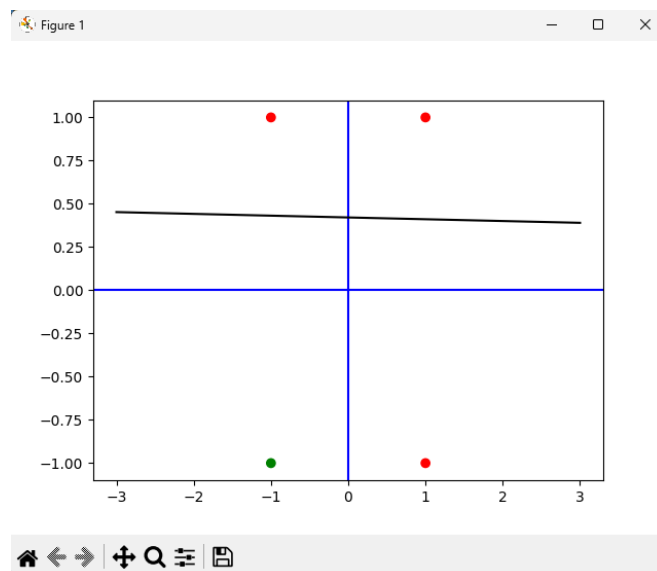
for i in range(columns - 1):
    x = np.array(np.loadtxt("entradas.csv", delimiter=',', usecols=i))
    arreglo.append(x)
X = np.array(arreglo)

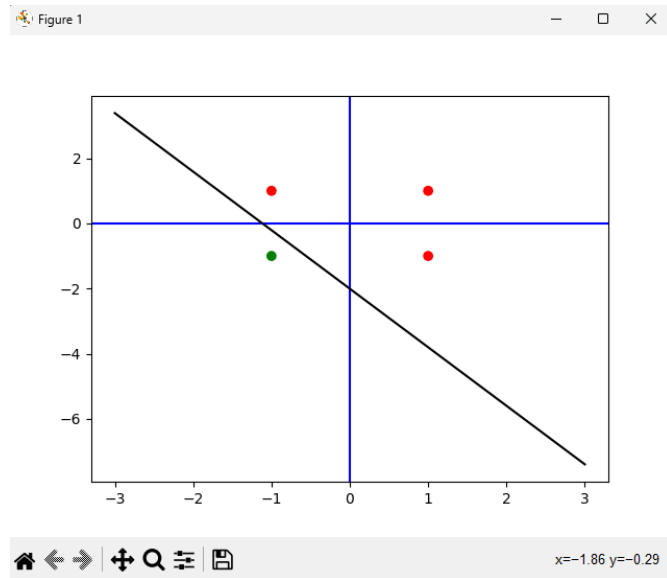
y = np.array(np.loadtxt("entradas.csv", delimiter=',', usecols=columns-1))

adaline.train(X, y, 10, 'grafica_', 0.95) # .95 -> 95% de precision

for i in range(rows):
    print(adaline.clasificar(X[:, i]))
```

Resultado obtenido:





```
klenb@AlanMtz200 MINGW64 /u/8º SEMESTRE/SEMINARIO IA 2/practicas/PRACTICA_3 (master)
$ python prueba.py
¡Precision del 95.0%!
[-0.53039058]
[-0.30947034]
[-0.13347238]
[0.08744785]
```

Conclusión:

Este código implementa el algoritmo de aprendizaje Adaline para clasificación de datos. La clase Adaline define un modelo neuronal con una cierta cantidad de entradas (dimensiones), una tasa de aprendizaje y una función de activación. El método train entrena el modelo en un conjunto de datos de entrada, utilizando un número determinado de épocas y una precisión objetivo. La función graph muestra la evolución del entrenamiento mediante la representación gráfica de los datos y la superficie de decisión. El código carga los datos desde un archivo CSV y entrena el modelo con ellos. Finalmente, el modelo se utiliza para clasificar los datos de entrada.

Código:

```
import numpy as np
import matplotlib.pyplot as plt
import csv
```

```

def graph(w_values):
    plt.clf()
    #Colorear entradas, si es 1 verde si es 0 rojo
    colors = np.where(y == 1, 'green', 'red')
    plt.scatter(X[0], X[1], color=colors)

    plt.axhline(color="blue")
    plt.axvline(color="blue")
    x_values = [-3,3]
    y_values = [-(adaline.w[0][0]/adaline.w[1][0])*(-3) - (adaline.b /
adaline.w[1][0]),
                -(adaline.w[0][0]/adaline.w[1][0])*(3) - (adaline.b /
adaline.w[1][0])]
    plt.plot(x_values, y_values, color="black")
    plt.pause(10)
    plt.close()

class Adaline:
    def __init__(self, dim, aprendizaje, activacion):
        self.n = dim
        self.aprendizaje = aprendizaje
        self.w = -1 + 2 * np.random.rand(dim, 1) # x = min + (max -
min)*rand()
        self.b = -1 + 2 * np.random.rand()
        self.activacion = activacion

    def clasificar(self, x):
        y = np.dot(self.w.transpose(), x) + self.b
        if self.activacion == "lineal":
            return y
        elif self.activacion == "logistica":
            return 1 / (1 + np.exp(-y))
        elif self.activacion == "sigmoidal":
            return np.tanh(y)

    def derivada_activacion(self, y):
        if self.activacion == "lineal":
            return 1
        elif self.activacion == "logistica":
            return y * (1 - y)
        elif self.activacion == "sigmoidal":
            return 1 - y ** 2

    def train(self, X, y, epocas, filename_prefix, precision=None):

```

```

n, m = X.shape
for i in range(epocas):
    # convertir los valores continuos de salida en valores binarios
    # comparar los vectores de salida de la red neuronal y los
    valores de la etiqueta de salida
    while not np.array_equal(np.sign(self.clasificar(X)),
np.sign(y)):
        for j in range(m):
            y_pred = self.clasificar(X[:, j])
            error = y[j] - y_pred
            delta = error * self.derivada_activacion(y_pred)
            self.w += self.aprendizaje * delta * X[:, j].reshape(-1,
1)

            self.b += self.aprendizaje * delta
        graph(self.w)
        if np.array_equal(np.sign(self.clasificar(X)), np.sign(y)):
            break
        # if precision is not None and
np.mean(np.allclose(self.clasificar(X), y)) >= float(precision):
        if precision is not None and
np.mean(np.sign(self.clasificar(X)) == np.sign(y)) >= float(precision): #
Con esta funcion al usar la activacion logistica la precision no suele dar
completamente 0 o 1
            print(f"¡Precision del {precision*100}%!")
            return
        plt.savefig(filename_prefix + str(i) + '.png')
        # Salir del bucle si se ha logrado una clasificación correcta
        if np.sign(self.clasificar(X)) == np.sign(y).all():
            print("¡Todas las clasificaciones son correctas!")
            break

# Obtener el numero de filas en la entrada del archivo
with open("entradas.csv") as file:
    rows = len(file.readlines())

# Obtener el numero de columnas en la entrada del archivo
with open("entradas.csv") as f:
    reader = csv.reader(f, delimiter=',')
    columns = len(next(reader))

adaline = Adaline(columns - 1, 0.1, "lineal") # activacion por defecto es
logistica
arreglo = []

```



```
for i in range(columns - 1):
    x = np.array(np.loadtxt("entradas.csv", delimiter=',', usecols=i))
    arreglo.append(x)
X = np.array(arreglo)

y = np.array(np.loadtxt("entradas.csv", delimiter=',', usecols=columns-1))

adaline.train(X, y, 10, 'grafica_', 0.95) # .95 -> 95% de precision

for i in range(rows):
    print(adaline.clasificar(X[:, i]))
```