Universidad de Guadalajara

Centro Universitario de Ciencias Exactas e Ingenierías

División de Electrónica y Computación

Departamento de Ciencias Computacionales

Ingeniería en Computación

Inteligencia Artificial II

Profesora: Arana Daniel, Nancy Guadalupe

I7040 – D01

Martes y Jueves 11:00 – 12:55

Actividad 02: Reporte de práctica del Adaline

Flores Camarena, Luis Manuel
214519661

Ojeda Escobar, César Arley
216306568

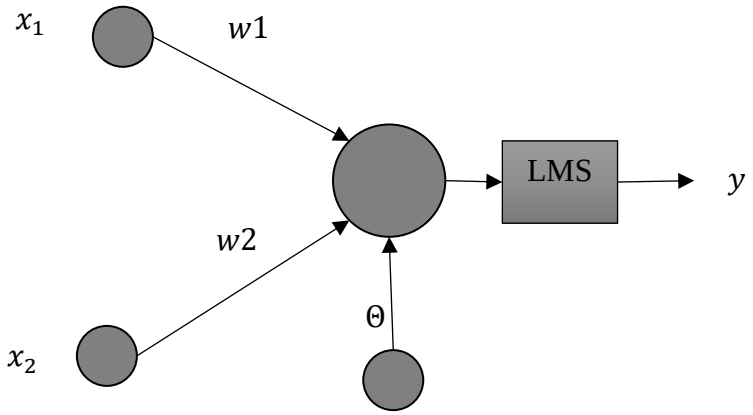
Fecha: 10/09/2019

El adaline

Este modelo introducido por Widrow en 1959, cuyo nombre proviene de ADAptative LINear Element, utiliza una neurona similar a la del perceptrón, pero de respuesta lineal cuyas entradas pueden ser continuas. Además, incorpora un parámetro adicional denominado net, que proporciona un grado de libertad adicional al algoritmo.

Descripción del modelo

La composición de esta neurona es muy parecida al perceptrón, pues esta basada en el mismo principio de entradas y pesos, sin embargo, utiliza un algoritmo de mínimos cuadrados la cual permite que la neurona se puede estar adaptando de manera continua a los datos de entrada, incluso después del entrenamiento.



En la figura anterior las entradas son x_1 y x_2, y la salida, y. Los pesos son $w1$ y $w2$. Además, existe un parámetro adicional llamada umbral y denotado por Θ. Por último tenemos la función de activación LMS (Least Mean Squares, mínimos cuadrados) la cual también es conocida como regla de Widrow-Hoff

Código

Se implemento el código para ser ejecutado en una plataforma web, siendo el frontend desarrollado con ayuda del framework React y el backend utilizando Python y Flask.

Algoritmo del Adaline en Python

Inicializamos la clase Adaline

```
class Adaline:
    def __init__(self, inputs, learningRate, expectedOutputs, maxEpochs, weights, theta, minError):
        self._inputs = inputs
        self._learningRate = learningRate
        self._expectedOutputs = expectedOutputs
        self._weights = np.array(weights)
        self._theta = theta
        self._outputs = np.zeros(inputs.shape[0])
        self._maxEpochs = maxEpochs
        self._minError = minError
        self.slope = []
        self.errors = []
        self.converged = False
```

Entrenamiento de la neurona

```
 def train(self):
     activated = self.activation()
     results = self._expectedOutputs - activated
     gradient = activated * (1-activated)
     error = (np.sum(np.array(results**2)))/2
     currentEpochs = 0

     if error < minError:
         self.graph(activated)
         self.errors.append(int(np.sum(results)))
         self.converged = True

     while currentEpochs < self._maxEpochs and error > minError:
         currentEpochs += 1
         i = 0
         self._theta = self._theta + (self._learningRate * results[i] * gradient[i] * 1)
         for inputs in self._inputs:
             self._weights = self._weights + (self._learningRate * results[i] * gradient[i] * inputs)
```

```
            i += 1
            if error < minError:
                self.converged = True
                break

        activated = self.activation()
        results = self._expectedOutputs - activated
        error = (np.sum(np.array(results**2)))/2
        self.errors.append(error)

        self.graph(activated)

    if error == 0 or error < minError:
        self.converged = True
```

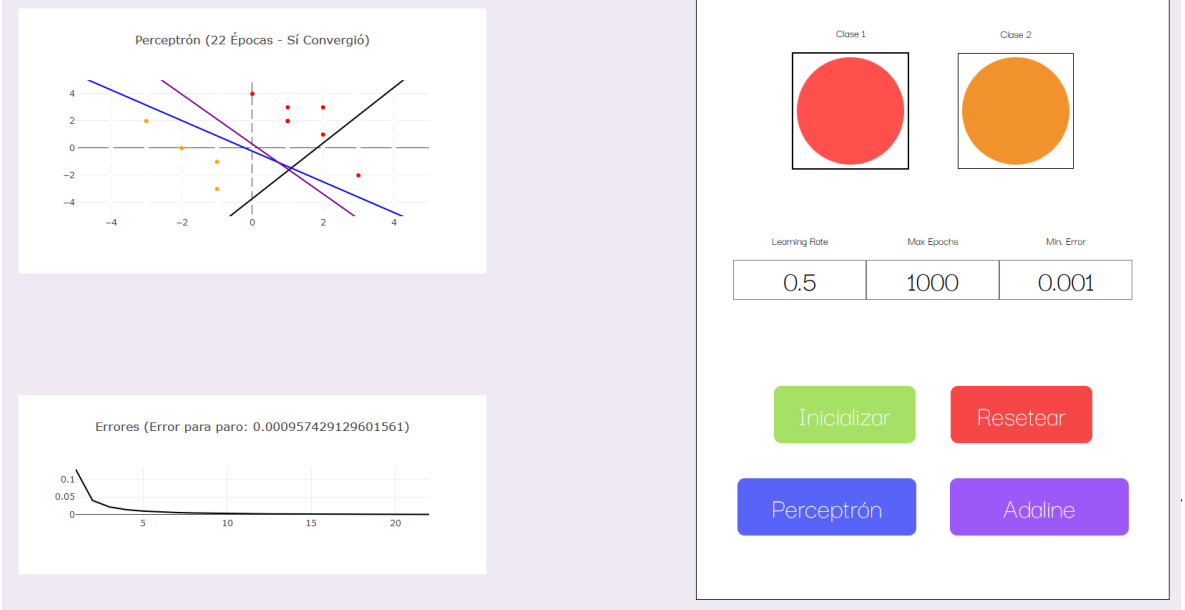
Función de activación

```
def activation(self):
    results = []
    dot = np.dot(self._inputs, self._weights) + self._theta

    for result in dot:
        results.append(1 / (1 + math.exp(-result)))

    r = np.array(results)
    r.sort()
    return np.array(results)
```

Interfaz de la aplicación



Conclusión

Diferencias entre el Adaline y el perceptrón

Podemos entonces concluir que la diferencia primordial entre la neurona Adaline y el Perceptrón es la capacidad que tiene la primera para poder seguir cambiando o adaptando los pesos para la clasificación de acuerdo con la entrada de nuevos valores. Este proceso es posible gracias a la implementación de la regla de Widrow-Hoff en el algoritmo del Adaline.

Sin embargo, su utilidad se ve limitada por tratarse de un sistema línea. Así, solo podrá separar correctamente patrones linealmente independientes, fallando en ocasiones ante patrones linealmente separables, que el perceptrón siempre discrimina. No obstante, ante patrones no separables linealmente, los resultados que proporciona son en promedio mejores que los del perceptrón, pues el Adaline siempre opera reduciendo el error cuadrático medio al mínimo posible [Bonifacio, 2002].

Por último, durante la ejecución de ambos algoritmos pudimos observar que, al haber mucho espacio en el plano, entre los puntos de las A y B, el **perceptrón** tiende a sesgarse a una de las clases, por ejemplo, a la clase A. Esto nos va a limitar a solo poder agregar valores por detrás del ultimo punto de la clase A, mientras que en la clase B (la que está lejos de la pendiente), se podrían introducir nuevos valores después del último punto. Por otro lado, en el algoritmo del **Adaline**, la recta de la pendiente tiene a posicionarse entre los dos puntos más próximos entre las dos clases, creando una especie de media aritmética, lo que nos permite agregar nuevos puntos de una clase, después del punto más cercano a la pendiente.