

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 04: Reporte de práctica del MLP con Levenberg- Marquardt

|  |  |
| --- | --- |
| Flores Camarena, Luis Manuel  214519661 | César Arley Ojeda Escobar  216306568 |

Fecha: 29/10/2019

# Levenberg-Marquardt

# Código

Se implemento una aplicación de escritorio con el lenguaje Python en su versión 3.7 y el módulo PyQt5 para el diseño de la interfaz.

## Algoritmo del Adaline en Python

### Inicializamos la clase Adaline

def \_\_init\_\_(self, entradas, clases, ratio\_aprendizaje, epocas\_maximas, pesos, theta, error\_minimo):

    self.ERROR\_MAX = 20

    self.entradas = np.array(entradas)

    self.salidas\_deseadas = np.array(clases)

    self.ratio\_aprendizaje = ratio\_aprendizaje

    self.epocas\_maximas = epocas\_maximas

    self.pesos = np.array(pesos)

    self.theta = theta

    self.error\_minimo = error\_minimo

    self.errores = []

### Entrenamiento de la neurona

def entrenamiento(self, ventana):

    progreso = ventana.progreso

    activacion = self.f\_activacion()

    resultados = self.salidas\_deseadas - activacion

    gradiente = activacion \* (1-activacion)

    # error = (np.sum(np.array(resultados\*\*2)))/len(self.pesos)

    error = (np.sum(np.array(resultados\*\*2)))/len(self.pesos)

    self.epoca\_actual = 0

    carga = 100 / self.epocas\_maximas

    if error < self.error\_minimo:

        self.errores.append(np.sum(resultados))

        self.cargarErrorGrafica(ventana)

        progreso.setValue(100)

        return True

    while self.epoca\_actual < self.epocas\_maximas and error > self.error\_minimo:

        self.epoca\_actual += 1

        i = 0

        progreso.setValue(self.epoca\_actual \* carga)

self.theta = self.theta + (self.ratio\_aprendizaje \* resultados[i] \* gradiente[i] \* 1)

        for i,entrada in enumerate(self.entradas):

            self.pesos = self.pesos + (self.ratio\_aprendizaje \* resultados[i] \* gradiente[i] \* entrada)

            if error < self.error\_minimo:

                progreso.setValue(100)

                return True

        activacion = self.f\_activacion()

        resultados = self.salidas\_deseadas - activacion

        gradiente = activacion \* (1-activacion)

        error = (np.sum(np.array(resultados\*\*2)))/len(self.pesos)

        self.errores.append(error)

        self.cargarErrorGrafica(ventana)

    if error == 0 or error < self.error\_minimo:

        progreso.setValue(100)

        return True

    return False

### Función de activación

def f\_activacion(self):

    resultados = []

    net = np.dot(self.entradas, self.pesos) + self.theta

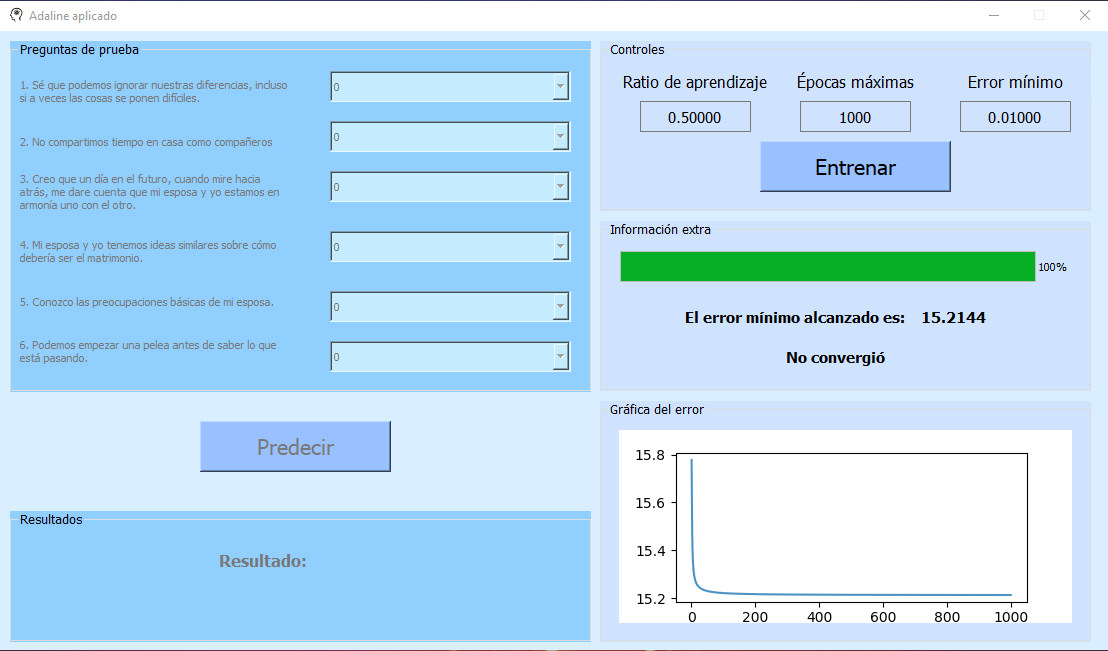
    # net = np.dot(self.entradas, self.pesos)

    for resultado in net:

        resultados.append(1 / (1 + math.exp(resultado\*-1)))

    return np.array(resultados)

# Interfaz de la aplicación

La interfáz se separa en dos partes, la parte de la derecha sirve para entrenar a la neurona, mientras que la parte de la derecha sirve para usar la neurona ya entrenada, sin embargo esta parte solo se activara cuando la neurona haya convergido correctamente.

## Data set de entrenamiento

# Conclusión

Es interesante poder aplicar nuestro conocimiento en situaciones reales de la vida y ver como los algoritmos que en un momento pueden parecer muy sencillos, puedan realmente ser aplicados en sistemas que puedan incluso hacer predicciones y como con un data set de entrenamiento de unos cuantos cientos de registros se puede tener una efectividad de hasta el 98.82% [Yöntem, 2018].

Sin embargo y a partir de varios entrenamientos con diferentes ratios de aprendizaje, épocas máximas y errores mínimos podemos concluir que nuestro set de entrenamiento no es linealmente separable, pues nunca se logró llegar a la convergencia de la neurona Adaline, por lo que habrá que esperar a implementar una red neuronal que nos permita clasificar problemas que quedan fuera del campo de los linealmente separables.

# Referencias