

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

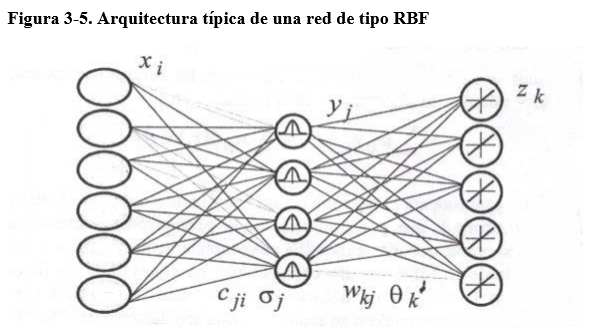
Actividad 06: Reporte de práctica de Clustering / Aproximación de Funciones

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

Fecha: 11/12/2019

# Introducción a las RBF

Las redes de base radial son un tipo de redes de neuronas artificiales que calculan la salida de la función en función de la distancia a un punto denominado centro. Al igual que con los perceptrones multicapa, sirven como aproximadores universales.



El aprendizaje consiste en la determinación de los centros, desviaciones y pesos de la capa oculta a la capa de salida. Como las capas de la red realizan diferentes tareas, se separarán los parámetros de la capa oculta de la capa de salida para optimizar el proceso. De esta forma, los centros y las desviaciones siguen un proceso guiado por una optimización en el espacio de entrada, mientras que los pesos siguen una optimización sobre la base de las salidas que se desean obtener.

# Código

def kmeans(self, X, hidden\_neurons):

        # Clustering

        # Choose random elements

        clusters = np.random.choice(np.squeeze(X), size=hidden\_neurons)

        prev\_clusters = clusters.copy()

        std\_dev = np.zeros(hidden\_neurons)

        converged = False

        while not converged:

            distances = np.squeeze(np.abs(X[:, np.newaxis] - clusters[np.newaxis, :]))

            closestCluster = np.argmin(distances, axis=1)

            for i in range(hidden\_neurons):

                pointsForCluster = X[closestCluster == i]

                if len(pointsForCluster) > 0:

                    clusters[i] = np.mean(pointsForCluster, axis=0)

            converged = np.linalg.norm(clusters - prev\_clusters) < 1e-6

            prev\_clusters = clusters.copy()

        distances = np.squeeze(np.abs(X[:, np.newaxis] - clusters[np.newaxis, :]))

        closestCluster = np.argmin(distances, axis=1)

        clustersWithNoPoints = []

        for i in range(hidden\_neurons):

            pointsForCluster = X[closestCluster == i]

            if len(pointsForCluster) < 2:

                clustersWithNoPoints.append(i)

                continue

            else:

                std\_dev[i] = np.std(X[closestCluster == i])

        if len(clustersWithNoPoints) > 0:

            pointsToAverage = []

            for i in range(hidden\_neurons):

                if i not in clustersWithNoPoints:

                    pointsToAverage.append(X[closestCluster == i])

            pointsToAverage = np.concatenate(pointsToAverage).ravel()

            std\_dev[clustersWithNoPoints] = np.mean(np.std(pointsToAverage))

        return clusters, std\_dev

def predict(self, X):

        y\_pred = []

        for i in range(X.shape[0]):

            outputs\_rbf = np.array([self.gaussian(X[i], c, s) for c, s, in zip(self.centroids, self.std\_dev)])

            net = outputs\_rbf.T.dot(self.w) + self.bias

            y\_pred.append(net)

        return np.array(y\_pred)

def gaussian(self, x, c, s):

        return np.exp(-1 / (2 \* s\*\*2) \* (x-c)\*\*2)

def fit(self, X, y):

        # Training hidden layer

        self.centroids, self.std\_dev = self.kmeans(X, self.hidden\_neurons)

        # Training output layer

        acumulated\_error = 999

        errors = []

        ephoc = 0

        while ephoc < self.max\_ephochs:

            self.acumulated\_outputs = []

            for i in range(X.shape[0]):

                outputs\_rbf = np.array([self.gaussian(X[i], c, s) for c, s, in zip(self.centroids, self.std\_dev)])

                net = outputs\_rbf.T.dot(self.w) + self.bias

                self.acumulated\_outputs.append(net)

                error = -(y[i] - net).flatten()

                errors.append(error)

                self.w = self.w - self.learning\_rate \* error \* outputs\_rbf

                self.bias = self.bias - self.learning\_rate \* error

            acumulated\_error = (np.sum(abs((np.array(y) - np.array(self.acumulated\_outputs))))) / (len(y)\*\*2)

            self.errors.append(acumulated\_error)

            ephoc += 1

# Conclusión

La aproximación mediante redes de base radial es un enfoque diferente al estándar que hemos viste con redes neuronales. Sin embargo, sigue siendo un aproximador universal, así como el perceptrón.

Sus aproximaciones son combinaciones lineales de múltiples funciones locales no lineales. Y esta clase de aproximación puede servir para el análisis de series, procesamiento de imágenes, reconocimiento del habla y hasta diagnósticos médicos.

# Referencias

Anónimo (2002). *Redes de Función de Base Radial*. Recuperado el 01 de noviembre del 2019 desde <http://decsai.ugr.es/~castro/Neuro/G-Funcionesdebaseradial/index.htm>]

Anónimo (2002 Redes de función de base radial (RBF). Recuperado el 01 de noviembre del 2019 desde <https://www.ibiblio.org/pub/linux/docs/LuCaS/Presentaciones/200304curso-glisa/redes_neuronales/curso-glisa-redes_neuronales-html/x185.html>]