Entrenamiento del perceptrón.

Perceptron training.

Sebastian Lopera Osorio, Luis David Restrepo Cadavid

[Sebastian.lopera@utp.edu.co](mailto:Sebastian.lopera@utp.edu.co), [david95rc@utp.edu.co](mailto:david95rc@utp.edu.co)

***Resumen-* *Se da una breve introducción a las redes neuronales, se describe el modelo del Perceptrón, historia, características, algoritmo de entrenamiento para resolver problemas de clasificación, se presentan también ejemplos resueltos, así como las limitantes de este tipo de red* [1]*.* *El algoritmo de entrenamiento del perceptrón se encuentra dentro de los denominados algoritmos por corrección de errores. Este tipo de algoritmos ajustan los pesos de manera proporcional a la diferencia entre la salida actual proporcionada por la red y la salida objetivo, con el fin de minimizar el error producido por la red.* [2]**

***Palabras clave-* Red neuronal, perceptrón, inteligencia artificial, entrenamiento, neurona.**

***Abstract*—We describe the Perceptron model, history, characteristics, training algorithm for solving classification problems, worked examples are also presented, as well as the limitations of this type of network. The perceptron training algorithm is within the so-called error correction algorithms. This type of algorithm adjusts the weights proportionally to the difference between the current output provided by the network and the target output in order to minimize the error produced by the network.**

***Keywords*- Neural network, perceptron, artificial intelligence, training, neuron.**

# INTRODUCCION

El concepto de redes neuronales suele ser explicado fácilmente cuando se relaciona con el aprendizaje humano. Cuando un niño va a aprender a identificar los colores, primero debe mirar el color y luego con indicaciones, aprende las características de ese color, esto le permite identificar cada uno de ellos en el futuro sin tener que recibir nuevamente la información. Lo mismo sucede con las redes neuronales, pueden tomar decisiones una vez han sido entrenadas con información previa. Si se desea que identifique colores como se mencionaba anteriormente, se debe suministrar información que permita identificar cada color tal cual como lo haría el niño en su aprendizaje.

En resumen, una red neuronal es la representación computacional de las neuronas del cerebro humano, las cuales intervienen en las decisiones que tomamos a diario y son claves al momento de adquirir conocimientos. Para comprender mejor el funcionamiento de una red neuronal, este tutorial se enfoca en el análisis del perceptrón puesto que es el modelo más simple de una red neuronal. [3]

# CONTENIDO

***El proceso de aprendizaje:***

En 1948 Norbert Wiener describió formalmente los principios de área de inteligencia en máquinas, a la cual llamó “cibernética” y desde entonces nace una especialidad conocida como ingeniería del conocimiento. El ingeniero de conocimiento busca la integración de un sistema computacional de tres elementos conocidos como representación, aprendizaje y razonamiento. Tradicionalmente, el estudio de la inteligencia artificial fue asociado por muchos años con el diseño de sistemas expertos, sistemas de juego como ajedrez y cualquier dispositivo que simulara los procesos cognitivos humanos. Sin embargo, estos sistemas basados en la representación simbólica del conocimiento pronto se vieron limitados para contribuir significativamente a la solución de problemas reales, en robótica, visión, reconocimiento de voz y otros. La razón fundamental, fue la limitación de estos sistemas para adquirir conocimiento de la manera en la que un organismo vivo con un sistemas nervioso central puede hacerlo. [4]

Como se mencionó anteriormente, la manera en que el cerebro humano evoluciona en su tiempo de vida es a través de un proceso conocido como aprendizaje. El producto del aprendizaje es lo que comúnmente conocemos como experiencias. El aprendizaje se realiza, en la mayoría de los casos, por ensayo y error. El cerebro genera una respuesta ante una determinada situación y una vez que ésta es aplicada, es posteriormente comparada con la respuesta deseada que proviene de una entidad ajenas al cerebro. Éste es el caso típico del bebé que está aprendiendo a hablar y que es corregido, una y otra vez, por sus padres en la forma correcta de pronunciar una determinada palabra. En ese caso tenemos una señal de entrada que, en nuestro ejemplo, sería la necesidad del bebé de producir una palabras y, finalmente, un profesor o maestro (los padres) que le dan al cerebro del bebé la retroalimentación que le permite ir modificando sus conexiones sinápticas para alcanzar, al cabo de meses, una pronunciación perfecta.

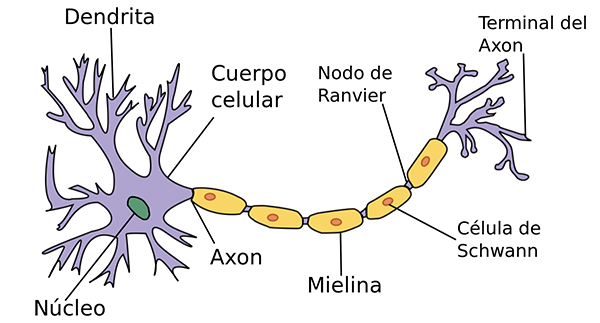
***Cómo entrenar un perceptrón:***

**¿qué es un perceptrón?**

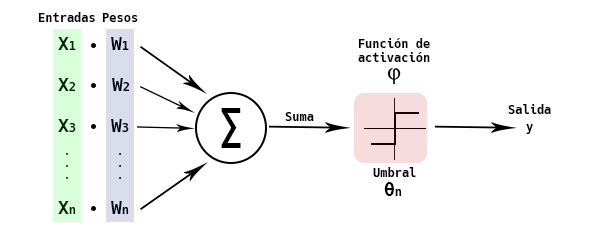
Cuántas veces habremos dicho u oído eso de…es que sólo tengo una neurona y no me da para más.

¿Y si fuera cierto? Bueno, algún que otro personaje público podría servirnos de ejemplo. Pero, bromas aparte, ¿qué podríamos hacer con una única neurona? Para averiguarlo, primero tendríamos que saber cómo funciona.

El modelo biológico de una neurona tiene esta pinta:

***Imagen 1.*** *[5]*

Resulta que en 1943, McCulloch y Pitts desarrollaron un modelo matemático que emula el comportamiento de una neurona. En el modelo biológico, las dentrinas tienen la función de recibir impulsos. En función de esos impulsos, se transmite una señal electroquímica a otras neuronas a través del Axón y sus terminales. La neurona McCulloch-Pitts (MCP) emula este comportamiento. El modelo recibe unas entradas, que llevan unos pesos asociados que simulan la intensidad de los impulsos. El valor de dichas señales se suma y, si dicha suma supera un cierto valor umbral, se produce una salida. El umbral actúa sobre una función de activación, que decidirá entre dos valores el valor de la salida, dependiendo si la suma de las señales supera o no dicho valor umbral.[5]



***Imagen 2.* [5]**





***Imagen 3.* [5]**

Haciendo una pequeña transformación podemos pasar el umbral θ al otro lado de la ecuación, lo que nos simplificará la vida. Si hacemos[5]



nos quedará:

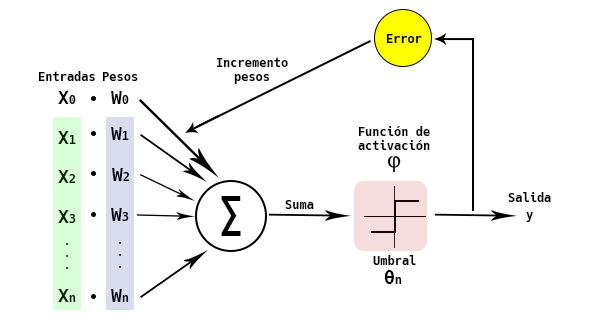




***Imagen 4.* [5]**

En 1957, Frank Rosenblatt puso nombre a lo que, en Machine Learning, podríamos llamar una neurona artificial. La llamó Perceptrón. Basándose en el modelo MCP, desarrolló un algoritmo matemático que, simulando el comportamiento de las neuronas, tiene la habilidad de aprender. Lo que consigue aprender dicho algoritmo son los pesos adecuados para que la salida sea la correcta. Y como al final está decidiendo entre dos valores, podemos decir que el perceptrón está clasificando en dos clases los datos.[5]

¿En qué consiste el algoritmo de Rosenblatt? Al perceptrón se le pasa un conjunto de datos de entrenamiento. La regla de aprendizaje del perceptrón consiste en ir añadiendo un incremento a los pesos de las entradas hasta que las salidas obtenidas por el perceptrón coincidan con las salidas esperadas del conjunto de datos de entrenamiento. Prueba y error.[5]



***Imagen 5.* [5]**

Al comienzo del algoritmo se escoge un valor igual a cero para todos los pesos.

A continuación, para todas entradas del conjunto de datos, se calcula la salida de la neurona y se incrementan los pesos. El incremento de los pesos se calcularía así:

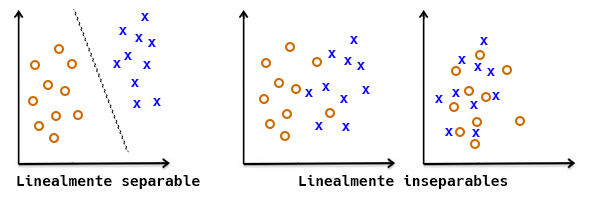


donde

* δ sería la salida esperada del conjunto de datos de entrenamiento
* y sería la salida calculada por la neurona.
* η sería la tasa de aprendizaje, con un valor entre 0 y 1.

Ojo!, que hay un par de peros a la hora de usar el perceptrón si queremos que converja, es decir, que logre calcular los pesos correctos y las salidas calculadas por la neurona coincidan con las salidas esperadas:

1.- El conjunto de datos tiene que ser linealmente separable.



***Imagen 6.* [5]**

2.- La tasa de aprendizaje tiene que ser suficientemente pequeña.[5]

Si no se cumplen las dos condiciones anteriores, nuestro perceptrón seguirá intentando calcular los pesos correctos hasta el infinito, y más allá.[5]

Bueno, hasta aquí la teoría. Ahora vamos a meternos en harina e intentar programar en Python un perceptrón que nos clasifique un conjunto de datos. Me ha sido difícil encontrar un dataset linealmente separable que no sea el típico Iris. Así que nos vamos a construir uno nosotros. Usaremos la librería de machine learning de Python: scikit-learn. Y más concretamente la función make blobs, para generar datos aleatorios para clasificación.[5]

Código en python para realizar el entrenamiento de un perceptrón:

//Se importan las librerías necesarias que permiten el entrenamiento

*import pandas as pd*

*import matplotlib.pyplot as plt*

*from sklearn.datasets import make\_blobs*

*from datetime import datetime*

*X, y = make\_blobs(n\_samples=1000,n\_features=2, centers=2, cluster\_std=1.5)*

*dictdata = dict(feature1=X[:,0], feature2=X[:,1], type=y)*

*df = pd.DataFrame(dictdata)*

*now = datetime.now()*

*filename = 'perceptron\_data\_%d%d%d.csv' %(now.year, now.month, now.day)*

*df.to\_csv(filename, sep = ';', index = False, encoding = 'utf-8')*

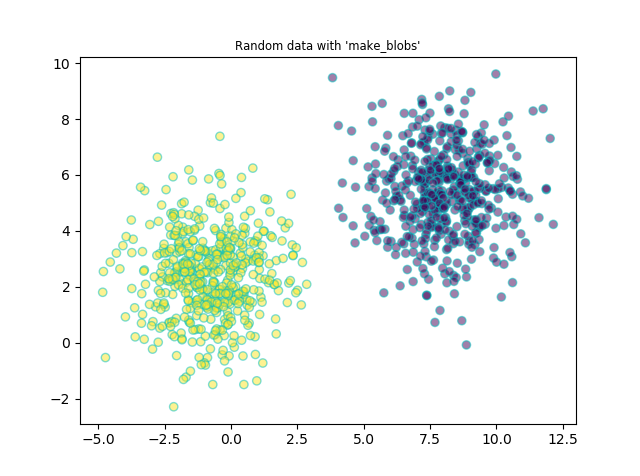
*plt.title("Random data with 'make\_blobs'", fontsize='small')*

*plt.scatter(X[:, 0], X[:, 1], marker='o', c=y, alpha=0.5, edgecolor='c')*

*plt.show()* [5]

Con éste código generamos un dataset con 1000 muestras, dos características y dos clusters, con una desviación típica de los clusters de 1.5. El código te hace una gráfica del dataset, donde se puede comprobar si son linealmente independientes. Además, guarda los datos en un archivo csv. Podéis probar hasta que el conjunto de datos devuelto por el código os convenza.[5]

Yo he elegido un dataset que tiene esta pinta, y podéis descargar aquí.



***Imagen 7.* [5]**

Vamos a echar un vistazo a los datos

*import pandas as pd*

*df = pd.read\_csv('perceptron\_data\_2018310.csv', sep=';')*

*print('\_'\*60 + 'COLUMNS')*

*print(df.columns.values)*

*print('\_'\*60 + 'INFO')*

*print (df.info())*

*print('\_'\*60 + 'DESCRIBE')*

*print (df.describe().transpose())*

*print('\_'\*60 + 'SHAPE')*

*print (df.shape)*

*print('\_'\*60 + 'COUNT VALUE CLASSES')*

*print (df.loc[:,'type'].value\_counts())*

*print('\_'\*60 + 'NULL VALUES')*

*print (df.isnull().sum())*

*import pandas as pd*

*df = pd.read\_csv('perceptron\_data\_2018310.csv', sep=';')*

*print('\_'\*60 + 'COLUMNS')*

*print(df.columns.values)*

*print('\_'\*60 + 'INFO')*

*print (df.info())*

*print('\_'\*60 + 'DESCRIBE')*

*print (df.describe().transpose())*

*print('\_'\*60 + 'SHAPE')*

*print (df.shape)*

*print('\_'\*60 + 'COUNT VALUE CLASSES')*

*print (df.loc[:,'type'].value\_counts())*

*print('\_'\*60 + 'NULL VALUES')*

*print (df.isnull().sum())* [5]

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_COLUMNS

['feature1' 'feature2' 'type'] [5]

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ INFO

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 1000 entries, 0 to 999

Data columns (total 3 columns):

feature1 1000 non-null float64

feature2 1000 non-null float64

type 1000 non-null int64

dtypes: float64(2), int64(1)

memory usage: 23.5 KB

None [5]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_DESCRIBE

count mean std min 25% 50% 75% \

feature1 1000.0 3.606442 4.639656 -4.831269 -0.778176 3.339908 7.970958

feature2 1000.0 3.883439 2.126527 -2.290611 2.287451 3.816354 5.482050

type 1000.0 0.500000 0.500250 0.000000 0.000000 0.500000 1.000000

max

feature1 12.142749

feature2 9.609528

type 1.000000 [5]

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_SHAPE

(1000, 3)

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_COUNT VALUE CLASSES

1 500

0 500

Name: type, dtype: int64 [5]

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_NULL VALUES

feature1 0

feature2 0

type 0

dtype: int64

[5]

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_COLUMNS

['feature1' 'feature2' 'type'] [5]

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_INFO

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 1000 entries, 0 to 999

Data columns (total 3 columns):

feature1 1000 non-null float64

feature2 1000 non-null float64

type 1000 non-null int64

dtypes: float64(2), int64(1)

memory usage: 23.5 KB

None

[5]

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ DESCRIBE

count mean std min 25% 50% 75% \

feature1 1000.0 3.606442 4.639656 -4.831269 -0.778176 3.339908 7.970958

feature2 1000.0 3.883439 2.126527 -2.290611 2.287451 3.816354 5.482050

type 1000.0 0.500000 0.500250 0.000000 0.000000 0.500000 1.000000

max

feature1 12.142749

feature2 9.609528

type 1.000000

[5]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_SHAPE

(1000, 3)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_COUNT VALUE CLASSES

1 500

0 500

Name: type, dtype: int64

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_NULL VALUES

feature1 0

feature2 0

type 0

dtype: int64

Ahora vamos a dividir nuestro dataset en dos conjuntos de datos, uno para el entrenamiento del perceptrón y otro para comprobar su correcto funcionamiento, con la función train\_test\_split. [5]

*from sklearn.model\_selection import train\_test\_split*

*X, y = df.loc[:, ['feature1', 'feature2']].values, df.loc[:,['type']].values*

*X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,test\_size=0.3)*

*from sklearn.model\_selection import train\_test\_split*

*X, y = df.loc[:, ['feature1', 'feature2']].values, df.loc[:,['type']].values*

*X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,test\_size=0.3)*

Y creamos una clase perceptrón que va a incluir tres métodos:

* zeta: con el que se calculan los productos de las entradas por sus pesos.
* predict: con el que se calcula la salida de la neurona a partir de los datos devueltos por zeta y teniendo en cuenta la función de activación. En la teoría hemos utilizado una función de activación que tenía dos estados, el 1 si z era mayor o igual a uno y -1 para el resto de casos. En nuestros datos de entrenamiento tenemos dos estados, dos clases, 1 y 0. Por lo tanto nuestra función de activación devolverá un 1 si z es mayor o igual a uno y 0 en el resto de casos.
* fit: donde se producen las iteraciones sobre el conjunto de los datos, contando el número de iteraciones, el número de errores en cada iteración y se van actualizando los pesos hasta llegar a cero errores. [5]

*class SimplePerceptron():*

*def \_\_init\_\_(self, eta):*

*"""*

*:param eta: tasa de aprendizaje*

*"""*

*self.eta = eta*

*def zeta(self, X):*

*"""*

*Calcula el producto de las entradas por sus pesos*

*:param X: datos de entrenamiento con las caracteristicas. Array*

*"""*

*zeta = np.dot(1, self.weights[0]) + np.dot(X, self.weights[1:])*

*return zeta*

*def predict(self, X):*

*"""*

*Calcula la salida de la neurona teniendo en cuenta la función de activación*

*:param X: datos con los que predecir la salida de la neurona. Array*

*:return: salida de la neurona*

*"""*

*output = np.where(self.zeta(X) >= 0.0, 1, 0)*

*return output*

*def fit(self, X, y):*

*#Ponemos a cero los pesos*

*self.weights = [0] \* (X.shape[1] + 1)*

*self.errors = []*

*self.iteraciones = 0*

*while True:*

*errors = 0*

*for features, expected in zip(X,y):*

*delta\_weight = self.eta \* (expected - self.predict(features))*

*self.weights[1:] += delta\_weight \* features*

*self.weights[0] += delta\_weight \* 1*

*errors += int(delta\_weight != 0.0)*

*self.errors.append(errors)*

*self.iteraciones += 1*

*if errors == 0:*

*break*

Vamos a probar el código. Creamos una instancia de la clase SimplePerceptron, y la entrenamos:

#*Creamos una instancia de la clase*

*sp = SimplePerceptron(eta=0.1)*

*#Entrenamos*

*sp.fit(X\_train, y\_train)*

*#Creamos una instancia de la clase*

*sp = SimplePerceptron(eta=0.1)*

*#Entrenamos*

*sp.fit(X\_train, y\_train)*

*Podemos plotear las iteraciones con sus errores:*

*Python*

*#Ploteamos las iteraciones y numero de errores*

*plt.plot(range(1, len(sp.errors) + 1), sp.errors, marker='o')*

*plt.xlabel('Iteracion')*

*plt.ylabel('Errores')*

*plt.tight\_layout()*

*plt.show()*

*#Ploteamos las iteraciones y numero de errores*

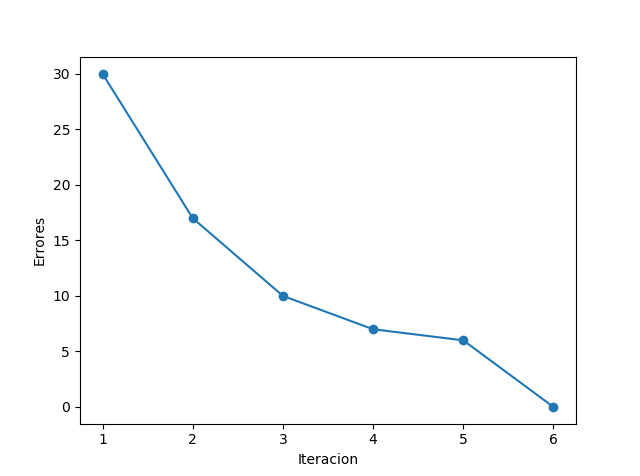
*plt.plot(range(1, len(sp.errors) + 1), sp.errors, marker='o')*

*plt.xlabel('Iteracion')*

*plt.ylabel('Errores')*

*plt.tight\_layout()*

*plt.show()* [5]



***Imagen 8.* [5]**

La gráfica muestra que ha necesitado 6 iteraciones sobre el conjunto de datos para calcular los pesos correctos.

Lo siguiente es pasarle a nuestro perceptrón entrenado el conjunto de datos de test y comprobar cuántos aciertos tiene:

#Comprobamos la precisión del perceptron con los datos de test [5]

*print('\_'\*60 + "Prediccion para X\_test")*

*prediction = sp.predict(X\_test)*

*print (prediction)*

*print('\_'\*60 + "Esperado para X\_test")*

*print (y\_test.T[0])*

*print('\_'\*60 + "¿Coincide lo esperado y lo devuelto por el perceptron?")*

*print (np.array\_equal(prediction, y\_test.T[0]))*

*print('\_'\*60 + "PRECISION")*

*print(str(np.sum(prediction == y\_test.T[0])/prediction.shape[0] \* 100) + ' %')*

*#Comprobamos la precisión del perceptron con los datos de test*

*print('\_'\*60 + "Prediccion para X\_test")*

*prediction = sp.predict(X\_test)*

*print (prediction)*

*print('\_'\*60 + "Esperado para X\_test")*

*print (y\_test.T[0])*

*print('\_'\*60 + "¿Coincide lo esperado y lo devuelto por el perceptron?")*

*print (np.array\_equal(prediction, y\_test.T[0]))*

*print('\_'\*60 + "PRECISION")*

*print(str(np.sum(prediction == y\_test.T[0])/prediction.shape[0] \* 100) + ' %')*

*TeX*

*[5]*

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ Prediccion para X\_test

[0 0 1 0 1 1 0 0 0 0 1 1 0 0 1 1 0 1 1 0 0 0 1 0 1 1 0 0 1 1 1 1 1 0 0 1 1

0 1 1 1 1 0 0 0 1 1 0 1 1 0 0 1 1 1 1 0 0 1 0 1 0 0 1 0 0 0 0 1 0 0 1 1 1

1 1 1 0 1 1 0 1 0 0 0 1 0 0 0 1 0 1 1 1 0 0 0 1 0 1 0 1 1 1 1 0 0 0 1 1 0

1 1 0 1 1 1 1 1 1 1 1 1 0 0 1 1 0 1 1 1 1 0 0 1 1 0 0 1 0 0 0 0 0 0 1 1 1

0 1 0 1 0 0 0 1 0 0 0 0 1 0 1 0 1 0 1 0 1 0 1 1 0 1 0 0 0 1 1 1 0 0 1 1 0

0 1 0 1 1 0 0 1 0 0 0 0 1 0 1 1 1 1 0 1 0 0 0 1 0 0 1 0 0 0 1 1 0 0 1 1 0

1 0 0 0 1 1 1 0 0 1 0 1 1 1 0 1 0 0 0 0 0 0 0 1 1 1 1 0 0 0 1 0 1 0 1 0 0

1 1 0 1 1 0 1 1 1 0 0 1 1 1 0 1 0 0 0 1 1 1 1 1 0 0 0 1 1 1 0 0 0 1 0 0 0

1 1 1 1]

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_Esperado para X\_test

[0 0 1 0 1 1 0 0 0 0 1 1 0 0 1 1 0 1 1 0 0 0 1 0 1 1 0 0 1 1 1 1 1 0 0 1 1

0 1 1 1 1 0 0 0 1 1 0 1 1 0 0 1 1 1 1 0 0 1 0 1 0 0 1 0 0 0 0 1 0 0 1 1 1

1 1 1 0 1 1 0 1 0 0 0 1 0 0 0 1 0 1 1 1 0 0 0 1 0 1 0 1 1 1 1 0 0 0 1 1 0

1 1 0 1 1 1 1 1 1 1 1 1 0 0 1 1 0 1 1 1 1 0 0 1 1 0 0 1 0 0 0 0 0 0 1 1 1

0 1 0 1 0 0 0 1 0 0 0 0 1 0 1 0 1 0 1 0 1 0 1 1 0 1 0 0 0 1 1 1 0 0 1 1 0

0 1 0 1 1 0 0 1 0 0 0 0 1 0 1 1 1 1 0 1 0 0 0 1 0 0 1 0 0 0 1 1 0 0 1 1 0

1 0 0 0 1 1 1 0 0 1 0 1 1 1 0 1 0 0 0 0 0 0 0 1 1 1 1 0 0 0 1 0 1 0 1 0 0

1 1 0 1 1 0 1 1 1 0 0 1 1 1 0 1 0 0 0 1 1 1 1 1 0 0 0 1 1 1 0 0 0 1 0 0 0

1 1 1 1]

¿Coincide lo esperado y lo devuelto por el perceptrón?

True

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_PRECISION

100.0 %

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ Prediccion para X\_test

[0 0 1 0 1 1 0 0 0 0 1 1 0 0 1 1 0 1 1 0 0 0 1 0 1 1 0 0 1 1 1 1 1 0 0 1 1

0 1 1 1 1 0 0 0 1 1 0 1 1 0 0 1 1 1 1 0 0 1 0 1 0 0 1 0 0 0 0 1 0 0 1 1 1

1 1 1 0 1 1 0 1 0 0 0 1 0 0 0 1 0 1 1 1 0 0 0 1 0 1 0 1 1 1 1 0 0 0 1 1 0

1 1 0 1 1 1 1 1 1 1 1 1 0 0 1 1 0 1 1 1 1 0 0 1 1 0 0 1 0 0 0 0 0 0 1 1 1

0 1 0 1 0 0 0 1 0 0 0 0 1 0 1 0 1 0 1 0 1 0 1 1 0 1 0 0 0 1 1 1 0 0 1 1 0

0 1 0 1 1 0 0 1 0 0 0 0 1 0 1 1 1 1 0 1 0 0 0 1 0 0 1 0 0 0 1 1 0 0 1 1 0

1 0 0 0 1 1 1 0 0 1 0 1 1 1 0 1 0 0 0 0 0 0 0 1 1 1 1 0 0 0 1 0 1 0 1 0 0

1 1 0 1 1 0 1 1 1 0 0 1 1 1 0 1 0 0 0 1 1 1 1 1 0 0 0 1 1 1 0 0 0 1 0 0 0

1 1 1 1]

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ Esperado para X\_test

[0 0 1 0 1 1 0 0 0 0 1 1 0 0 1 1 0 1 1 0 0 0 1 0 1 1 0 0 1 1 1 1 1 0 0 1 1

0 1 1 1 1 0 0 0 1 1 0 1 1 0 0 1 1 1 1 0 0 1 0 1 0 0 1 0 0 0 0 1 0 0 1 1 1

1 1 1 0 1 1 0 1 0 0 0 1 0 0 0 1 0 1 1 1 0 0 0 1 0 1 0 1 1 1 1 0 0 0 1 1 0

1 1 0 1 1 1 1 1 1 1 1 1 0 0 1 1 0 1 1 1 1 0 0 1 1 0 0 1 0 0 0 0 0 0 1 1 1

0 1 0 1 0 0 0 1 0 0 0 0 1 0 1 0 1 0 1 0 1 0 1 1 0 1 0 0 0 1 1 1 0 0 1 1 0

0 1 0 1 1 0 0 1 0 0 0 0 1 0 1 1 1 1 0 1 0 0 0 1 0 0 1 0 0 0 1 1 0 0 1 1 0

1 0 0 0 1 1 1 0 0 1 0 1 1 1 0 1 0 0 0 0 0 0 0 1 1 1 1 0 0 0 1 0 1 0 1 0 0

1 1 0 1 1 0 1 1 1 0 0 1 1 1 0 1 0 0 0 1 1 1 1 1 0 0 0 1 1 1 0 0 0 1 0 0 0

1 1 1 1]

¿Coincide lo esperado y lo devuelto por el perceptron?

True

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_PRECISION

100.0 %

¡Magia!, tenemos un 100% de acierto sobre nuestros datos de test!

1. CONCLUSIONES:

•El perceptrón, a pesar de ser una de las redes más utilizadas, no es una de las más potentes ya que posee ciertas limitaciones, por ejemplo, el caso del aprendizaje en problemas complejos.

•Este tipo de redes se pueden implementar en la vida moderna en ámbitos como análisis de series temporales, procesamiento de imágenes, reconocimiento automático del habla, diagnósticos médicos, entre otros. [6]

1. Referencias
2. <https://www.uaeh.edu.mx/docencia/P_Presentaciones/huejutla/sistemas/redes_neuronales/perceptron.pdf>
3. <http://laboratorios.fi.uba.ar/lsi/bertona-tesisingenieriainformatica.pdf>
4. <https://platzi.com/tutoriales/1157-ia-2017/2619-entrenamiento-del-perceptron/>

1. [https://books.google.com.co/books?id=iCiiyyCFZuIC&pg=PA80&dq=entrenamiento+de+un+perceptron&hl=es&sa=X&ved=0ahUKEwiEgrTO55TkAhUkzlkKHewWBeoQ6AEINDAC#v=onepage&q=entrenamiento%20de%20un%20perceptron&f=false](https://books.google.com.co/books?id=iCiiyyCFZuIC&pg=PA80&dq=entrenamiento+de+un+perceptron&hl=es&sa=X&ved=0ahUKEwiEgrTO55TkAhUkzlkKHewWBeoQ6AEINDAC" \l "v=onepage&q=entrenamiento%20de%20un%20perceptron&f=false)
2. [HTTPS://KOLDOPINA.COM/COMO-ENTRENAR-A-TU-PERCEPTRON/](https://koldopina.com/como-entrenar-a-tu-perceptron/)
3. <https://es.slideshare.net/Jeffo92/perceptrn-simple-y-multicapa>