Universidad de Guadalajara

Centro Universitario de Ciencias Exactas e Ingenierías
División de Tecnologías para la Integración Ciber-Humana

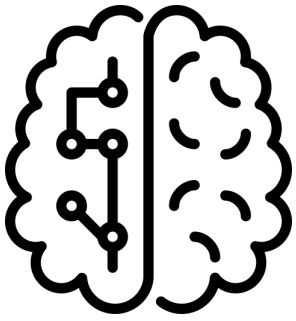
Seminario de Solución de Problemas de Inteligencia Artificial II

Ingeniería en Computación
Sección D05

Practica 1 – Ejercicio 1

Perceptrón Simple

Carrillo Partida Jorge Alberto
216439258
Jorge.carrillo4392@alumnos.udg.mx



Profesores
M. Diego Alberto Oliva Navarro
M. Diego Campos Peña

26 de Febrero de 2024

Perceptrón Simple

Introducción

Presentamos el desarrollo y la evaluación de un perceptrón simple, una forma básica de red neuronal artificial. Este trabajo tiene como objetivo explorar la implementación y funcionalidad del perceptrón en tareas de clasificación binaria, utilizando específicamente las compuertas XOR y OR para probar el modelo. Se detalla la creación de un programa que no solo lee y procesa patrones de entrenamiento desde un archivo, sino que también permite la configuración de varios parámetros de entrenamiento, incluyendo la tasa de aprendizaje y el número máximo de épocas. A través de este ejercicio, buscamos demostrar cómo el perceptrón simple puede abordar problemas de clasificación complejos.

Desarrollo

Código en Python:

```python
# Universidad de Guadalajara
# Centro Universitario de Ciencias Exactas e Ingenierías
# División de Tecnologías para la Integración Ciber-Humana
#
# Seminario de Solución de Problemas de Inteligencia Artificial II
# Ingenieria en Computación
# Sección D05
#
# Práctica 1 - Ejercicio 1
#
# Jorge Alberto Carrillo Partida / 216439258 / jorge.carrillo4392@alumnos.udg.mx
#
# M. Diego Campos - 26 de Febrero de 2024

# ==============================================================================

# Librerías necesarias para el funcionamiento del programa
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.utils import shuffle

# Clase necesaria para la red neuronal
class Perceptron:
    def __init__(self, learning_rate, epochs):
        self.learning_rate = learning_rate  # Tasa de aprendizanje
        self.epochs = epochs                # Número de epocas
        self.weights = None                 # Inicialización de pesos
        self.bias = 0                       # Inicialización de sesgo

    # Función de activación
    def activation(self, x):
        return np.where(x >= 0, 1, -1)

    # Método para entrenar el perceptron
    def fit(self, X, y):
        n_samples, n_features = X.shape
        self.weights = np.zeros(n_features)
        self.bias = 0

        for _ in range(self.epochs):
            for idx, x_i in enumerate(X):
                linear_output = np.dot(x_i, self.weights) + self.bias
                y_predicted = self.activation(linear_output)
                update = self.learning_rate * (y[idx] - y_predicted)
                self.weights += update * x_i
                self.bias += update

    # Método para realizar predicciones sobre los nuevos datos
    def predict(self, X):
        linear_output = np.dot(X, self.weights) + self.bias
        return self.activation(linear_output)

    # Método para visualizar la frontera de decisión del perceptron
    def decision_boundary(self, X, y):
        fig, ax = plt.subplots()
        ax.scatter(X[:, 0], X[:, 1], c=y, cmap='coolwarm', marker='o', label='Training data')
        x1 = np.linspace(X[:, 0].min(), X[:, 0].max(), 100)
        x2 = -(self.weights[0] * x1 + self.bias) / self.weights[1]
        ax.plot(x1, x2, 'k', label='Frontera de decisión')
        ax.set_xlabel('X1')
        ax.set_ylabel('X2')
        ax.legend()
        plt.title('Frontera de decisión del Perceptron - XOR')
        plt.show()

# Cargar los datos de los documentos de la práctica
data_trn = pd.read_csv('XOR_trn.csv')
data_tst = pd.read_csv('XOR_tst.csv')

# Combinar, mezclar y dividir los datos para un entrenamiento con el 80% de datos y 20% para las pruebas
combined_X = np.vstack((data_trn[['X1', 'X2']].values, data_tst[['X1', 'X2']].values))
combined_y = np.hstack((data_trn['Y'].values, data_tst['Y'].values))
combined_X, combined_y = shuffle(combined_X, combined_y, random_state=1)
```

```
split_index = int(0.8 * len(combined_y))
X_train = combined_X[:split_index]
y_train = combined_y[:split_index]
X_test = combined_X[split_index:]
y_test = combined_y[split_index:]

# Crear, entrenar y evaluar el perceptrón
perceptron = Perceptron(learning_rate=0.1, epochs=10)
perceptron.fit(X_train, y_train)
predictions = perceptron.predict(X_test)
accuracy = np.mean(predictions == y_test)
print(f'Accuracy: {accuracy * 100:.2f}%')

# Visualizar la frontera de decisión
perceptron.decision_boundary(X_train, y_train)
```

Código en partes:

1. Librerías

```
# Librerías necesarias para el funcionamiento del programa
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.utils import shuffle
```

El script inicia importando las librerías necesarias para su ejecución: ***numpy***, ***matplotlib.pyplot***, ***pandas*** y ***shuffle*** de ***sklearn.utils***. ***numpy*** es utilizado para operaciones matemáticas y manejo de arrays; ***matplotlib.pyplot*** permite la visualización de datos, en este caso, para graficar la frontera de decisión del perceptrón; ***pandas*** se usa para la carga y manipulación de datos, y ***shuffle*** ayuda a mezclar los datos para garantizar que la división entre los datos de entrenamiento y prueba sea aleatoria y equitativa.

2. Perceptron y entrenamiento

```
# Clase necesaria para la red neuronal
class Perceptron:
    def __init__(self, learning_rate, epochs):
        self.learning_rate = learning_rate  # Tasa de aprendizanje
        self.epochs = epochs                # Número de epocas
        self.weights = None                 # Inicialización de pesos
        self.bias = 0                       # Inicialización de sesgo

    # Función de activación
    def activation(self, x):
        return np.where(x >= 0, 1, -1)

    # Método para entrenar el perceptron
    def fit(self, X, y):
        n_samples, n_features = X.shape
        self.weights = np.zeros(n_features)
        self.bias = 0

        for _ in range(self.epochs):
            for idx, x_i in enumerate(X):
                linear_output = np.dot(x_i, self.weights) + self.bias
                y_predicted = self.activation(linear_output)
                update = self.learning_rate * (y[idx] - y_predicted)
                self.weights += update * x_i
                self.bias += update

    # Método para realizar predicciones sobre los nuevos datos
    def predict(self, X):
        linear_output = np.dot(X, self.weights) + self.bias
        return self.activation(linear_output)
```

Se define una clase **Perceptron**, cuyo constructor recibe dos parámetros: la **tasa de aprendizaje** y el ***número de épocas***. Dentro de esta clase, ***weights*** y ***bias*** son inicializados; ***weights*** es un array que almacenará los pesos asociados a las características de los inputs, y ***bias*** es un término que se añade para ajustar la salida del modelo además de la combinación lineal de inputs y pesos. El método ***fit*** de la clase ***Perceptron*** es responsable del entrenamiento del modelo. Aquí, cada muestra en el conjunto de entrenamiento es usada para ajustar los pesos y el sesgo basándose en la salida predicha y la real.

Después del entrenamiento, el método ***predict*** se utiliza para evaluar nuevas muestras y predecir su clase basándose en el modelo entrenado. Este método calcula la salida lineal utilizando los pesos y el sesgo ajustados, y luego aplica la función de activación para clasificar la entrada.

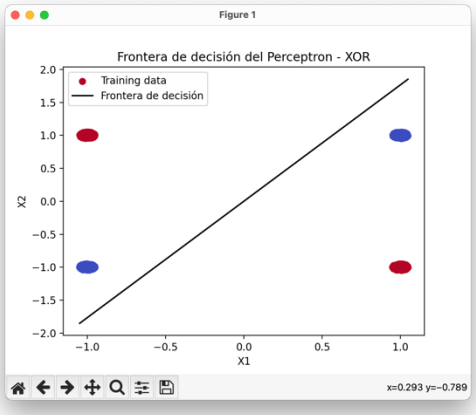
Lectura de datos y generación de resultados

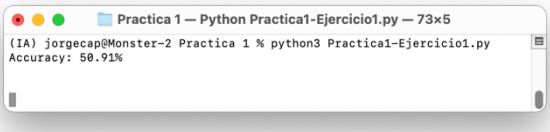
```
    # Método para visualizar la frontera de decisión del perceptron
    def decision_boundary(self, X, y):
        fig, ax = plt.subplots()
        ax.scatter(X[:, 0], X[:, 1], c=y, cmap='coolwarm', marker='o', label='Training data')
        x1 = np.linspace(X[:, 0].min(), X[:, 0].max(), 100)
        x2 = -(self.weights[0] * x1 + self.bias) / self.weights[1]
        ax.plot(x1, x2, 'k', label='Frontera de decisión')
        ax.set_xlabel('X1')
        ax.set_ylabel('X2')
        ax.legend()
        plt.title('Frontera de decisión del Perceptron - XOR')
        plt.show()

# Cargar los datos de los documentos de la práctica
data_trn = pd.read_csv('XOR_trn.csv')
data_tst = pd.read_csv('XOR_tst.csv')

# Combinar, mezclar y dividir los datos para un entrenamiento con el 80% de datos y 20% para las pruebas
combined_X = np.vstack((data_trn[['X1', 'X2']].values, data_tst[['X1', 'X2']].values))
combined_y = np.hstack((data_trn['Y'].values, data_tst['Y'].values))
combined_X, combined_y = shuffle(combined_X, combined_y, random_state=1)

split_index = int(0.8 * len(combined_y))
X_train = combined_X[:split_index]
y_train = combined_y[:split_index]
X_test = combined_X[split_index:]
y_test = combined_y[split_index:]

# Crear, entrenar y evaluar el perceptrón
perceptron = Perceptron(learning_rate=0.1, epochs=10)
perceptron.fit(X_train, y_train)
predictions = perceptron.predict(X_test)
accuracy = np.mean(predictions == y_test)
```

El script carga dos conjuntos de datos, uno para entrenamiento y otro para pruebas, que se combinan, mezclan y finalmente se dividen según una proporción de 80% para entrenamiento y 20% para pruebas, utilizando las funciones de ***pandas*** y ***shuffle***. Una vez que el modelo está entrenado y evaluado, la precisión del perceptrón se calcula como el porcentaje de predicciones correctas en el conjunto de pruebas. Finalmente, se visualiza la frontera de decisión usando ***matplotlib***, mostrando cómo el perceptrón ha aprendido a separar las dos clases en el espacio de características.

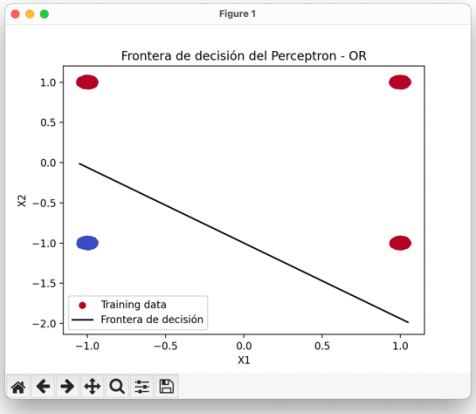
Resultados

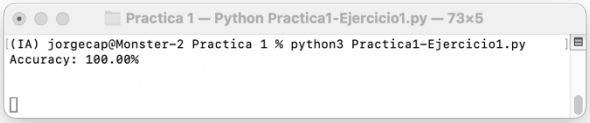
Resultados de la compuerta XOR:





Resultados de la compuerta OR:





Conclusión

La aplicación del perceptrón simple al problema de la compuerta XOR ha sido una experiencia de aprendizaje muy valiosa y práctica. En estan actividad, he implementado con éxito un sistema básico de aprendizaje automático que, a pesar de su simplicidad, me ha permitido entender de manera clara los fundamentos de la clasificación binaria y del aprendizaje de un perceptrón simple. Aunque el perceptrón simple tiene sus limitaciones con problemas complejos y no linealmente separables, como la compuerta XOR, también demostró ser eficaz en situaciones más sencillas, como lo fue con la compuerta OR. Esta actividad no solo ha profundizado en el conocimiento teórico sobre los perceptrones simples, si no que, al ponerlo a prueba fue mucho más facil entender su funcionamiento y como trabaja con los datos.