Dibujo de una persona

Descripción generada automáticamente con confianza bajaInstituto Politécnico Nacional

Escuela Superior de Cómputo

**Integrante:**

Torres Abonce Luis Miguel

**Grupo:**

6CV1

**Materia:**

Machine Learning

**Profesora:**

Camacho Vázquez Vanessa Alejandra

**Practica 9**

Aplicando retro propagación en Python

**Introducción**

La retropropagación (backpropagation) es un algoritmo esencial para el entrenamiento de redes neuronales artificiales. Introducido por David E. Rumelhart y otros en 1986, la retropropagación revolucionó el campo de la inteligencia artificial al permitir el ajuste eficiente de los pesos en redes neuronales de múltiples capas. Este algoritmo se utiliza para minimizar la función de pérdida de la red ajustando los pesos y sesgos mediante el cálculo de los gradientes. La práctica se centra en implementar la retropropagación en Python, empleando bibliotecas como NumPy, para entrenar una red neuronal simple con una capa oculta que utiliza la función de activación sigmoide. El objetivo es comprender el proceso de entrenamiento de una red neuronal y observar cómo se optimizan los parámetros para mejorar la precisión del modelo.

Desarrollo

Implementación de la Red Neuronal:

1. Inicialización de Parámetros: Se definen los pesos y sesgos de la red neuronal. La red cuenta con una capa oculta y una capa de salida.
2. Propagación Hacia Adelante (Forward Propagation): Se calcula la salida de la red utilizando la función de activación sigmoide. Esto implica calcular la salida de la capa oculta y luego la salida final.
3. Cálculo del Error: Se calcula el error como la diferencia entre la salida real y la salida predicha.
4. Retropropagación (Backpropagation): Se calculan los gradientes de la función de pérdida con respecto a los pesos y sesgos utilizando la regla de la cadena. Estos gradientes se utilizan para actualizar los pesos y sesgos de la red.
5. Entrenamiento: El proceso de propagación hacia adelante y retropropagación se repite durante un número fijo de épocas o hasta que se alcance un cierto criterio de convergencia.

**Código:**

import numpy as np

# Función de activación sigmoide

def sigmoid*(x)*:

    return 1 / (1 + np.exp(-x))

# Derivada de la función de activación sigmoide

def sigmoid\_derivative*(x)*:

    return x \* (1 - x)

# Datos de entrada y salida

X = np.array([[0, 0],

    [0, 1],

    [1, 0],

    [1, 1]])

y = np.array([[0],

    [1],

    [1],

    [0]])

# Inicialización de pesos y bias

np.random.seed(1)

input\_neurons = 2

hidden\_neurons = 3

output\_neurons = 1

weights\_input\_hidden = np.random.uniform(size=(input\_neurons, hidden\_neurons))

weights\_hidden\_output = np.random.uniform(size=(hidden\_neurons, output\_neurons))

bias\_hidden = np.random.uniform(size=(1, hidden\_neurons))

bias\_output = np.random.uniform(size=(1, output\_neurons))

# Hiperparámetros

epochs = 10000

learning\_rate = 0.1

# Entrenamiento

for epoch in range(epochs):

    # Feedforward

    input\_hidden = np.dot(X, weights\_input\_hidden) + bias\_hidden

    output\_hidden = sigmoid(input\_hidden)

    input\_output = np.dot(output\_hidden, weights\_hidden\_output) + bias\_output

    output = sigmoid(input\_output)

# Retropropagación

# Calcular el error

error = y - output

#Calcular los deltas y ajustar los pesos

delta\_output = error \* sigmoid\_derivative(output)

error\_hidden = delta\_output.dot(weights\_hidden\_output.T)

delta\_hidden = error\_hidden \* sigmoid\_derivative(output\_hidden)

# Actualizar pesos y bias

weights\_hidden\_output += output\_hidden.T.dot(delta\_output) \* learning\_rate

bias\_output += np.sum(delta\_output, axis=0, keepdims=True) \* learning\_rate

weights\_input\_hidden += X.T.dot(delta\_hidden) \* learning\_rate

bias\_hidden += np.sum(delta\_hidden, axis=0, keepdims=True) \* learning\_rate

#Resultados

print("Resultado después del entrenamiento:")

print(output)

**Resultado:**

Texto

Descripción generada automáticamente

**Conclusión**

La implementación de la retropropagación en Python utilizando NumPy nos permite entender y aplicar uno de los algoritmos más importantes en el campo del aprendizaje automático y las redes neuronales. Al ajustar los pesos y sesgos de la red para minimizar el error, podemos entrenar efectivamente una red neuronal para realizar tareas de clasificación. Esta práctica no solo refuerza los conceptos teóricos de la retropropagación, sino que también proporciona una base sólida para futuros estudios y aplicaciones en inteligencia artificial. Los resultados obtenidos demuestran la capacidad del algoritmo para aprender y mejorar su desempeño a través de iteraciones, destacando la importancia de la retropropagación en el entrenamiento de redes neuronales.