Informe de Clasificador No Lineal (Red Neuronal con Una Capa Oculta)

Descripción General

Este informe documenta la implementación de una red neuronal desde cero en Python, diseñada como un clasificador no lineal con una capa oculta. Esta red fue implementada en el contexto de una evaluación de Computación Blanda y aplicada a un problema de clasificación multiclase.

Estructura de la Red

La red consta de:

- Una **capa de entrada** con dos neuronas (correspondiente a las coordenadas X, Y de entrada).
- Una **capa oculta** con 10 neuronas y función de activación ReLU.
- Una **capa de salida** con 3 neuronas (una por clase: purple, orange, green), con activación Softmax.

Inicialización de Pesos

Se emplea la inicialización de He:

$\ w \sim \mathcal{N}(0, \sqrt{\frac{2}{n}}) $

Donde n es el número de entradas a la capa. Esta técnica mejora la convergencia al entrenar redes profundas con ReLU.

Funciones de Activación

ReLU (Rectified Linear Unit)

$\ \text{ReLU}(x) = \max(0, x) $

La derivada usada en el backpropagation es:

$
\text{ReLU}'(x) = \begin{cases} 1 & \text{si } x > 0 \ 0 & \text{si } x \leq 0 \end{cases} $

Softmax

$
\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}} $

Convierte la salida de la última capa en probabilidades.

Cálculo de la Pérdida

Se utiliza la función de pérdida **log loss** para clasificación multiclase:

$
\mathcal{L}(y, p) = - y \log(p) - (1 - y) \log(1 - p) $

Donde y es el vector objetivo y p la predicción.

Propagación Hacia Adelante (Forward Pass)

1. Z1 = X * W1 + b1
2. A1 = ReLU(Z1)
3. Z2 = A1 * W2 + b2
4. A2 = Softmax(Z2) => Predicciones

Retropropagación (Backpropagation)

Se actualizan los pesos usando el descenso por gradiente estocástico:

1. Error en salida:
   $\ dZ2 = A2 - Y $ $\ dW2 = A1^T \cdot dZ2 $ $\ db2 = \sum dZ2 $
2. Error en capa oculta:
   $\ dZ1 = (dZ2 \cdot W2) \odot \text{ReLU}'(Z1) $ $\ dW1 = X^T \cdot dZ1 $ $\ db1 = \sum dZ1 $
3. Actualización de pesos:
   $\ W = W - \eta \cdot dW $
   $\ b = b - \eta \cdot db $

Donde \(\eta \) es la tasa de aprendizaje.

Entrenamiento

- Se realiza por **epoch**, iterando sobre los datos en batches de tamaño 1 (SGD).
- En cada iteración, se calcula el error y se actualizan los pesos.

Resultados

Tras 1000 iteraciones de entrenamiento con tasa de aprendizaje 0.01, la red fue capaz de generalizar sobre un conjunto de prueba con entradas no vistas, clasificando con buena precisión las clases purple, orange y green.

Conclusión

La red neuronal implementada cumple con las características de un clasificador no lineal con una capa oculta. Utiliza ReLU, softmax, inicialización de He y entrenamiento mediante descenso por gradiente estocástico, cubriendo los fundamentos teóricos requeridos por la cátedra de Computación Blanda.

Archivos Implementados

`RN.py`

Contiene la clase `CBNN`, con la arquitectura de la red neuronal, funciones de activación, forward pass, backpropagation y entrenamiento.

`punto1.py`

Define los datos de entrada y objetivos, instancia la red y ejecuta el entrenamiento y pruebas.