Neural Network Code Report

Vihan Vashishth - IMT2021065 Shishir Shahi - IMT2021029 Allu Pavankarthik Reddy - IMT2021026 Rithik Bansal - IMT2021099

1. Introduction

The provided code implements a simple neural network framework in Java. The neural network consists of input, hidden, and output layers, with functionality for training using backpropagation. The code employs the OpenCSV library for reading input data from CSV files.

2. Structure and Components

a. NeuralNetwork Class
   i. Static Variables: layers and tDataSet are static arrays representing neural network layers and training data, respectively.
   ii. Main Method: Initializes the neural network layers, sets weight ranges, creates training data, and executes the training loop.
   iii. Training Loop: Iterates through training data, performs forward and backward propagation, and updates weights.

b. Layer Class
   i. Attributes: neurons array holds neuron instances for the layer.
   ii. Constructors:
      1. Hidden/Output Layer: Accepts the number of neurons and the number of weights for each neuron, initializes neurons with random weights and biases.
      2. Input Layer: Accepts input data, initializes neurons without weights or biases.

c. Neuron Class
   i. Attributes: weights, cache_weights, gradient, bias, and value represent neuron properties.
   ii. Constructors:
      1. Hidden/Output Neuron: Accepts weights and bias, initializes with cache weights and zero gradient.
      2. Input Neuron: Accepts input value, no weights or biases.
      3. Static Methods: setRangeWeight sets min and max weight values.
      4. Update Weight: Method updates neuron weights with cached weights.

d. StatUtil Class
    i. Methods:
        1. RandomFloat: Generates random float within a specified range.
        2. Sigmoid: Computes the sigmoid function.
        3. SigmoidDerivative: Calculates the derivative of the sigmoid function.
        4. ReLU: Computes the Rectified Linear Unit (ReLU) function.
        5. ReLUDerivative: Calculates the derivative of the ReLU function.
        6. squaredError: Computes the squared error between output and target.
        7. sumSquaredError: Computes the sum of squared errors (currently unused).

e. TrainingData Class
    i. Attributes: data and expectedOutput represent input data and corresponding expected output.
    ii. Constructor: Accepts input data and expected output.

3. Training Process

a. Initialization:
    i. Sets weight ranges.
    ii. Creates layers (input, hidden, and output) with specified neuron configurations.
    iii. Initializes training data.

b. Forward Propagation:
    i. Computes output for each layer using the sigmoid activation function.
    ii. Backward Propagation:
    iii. Updates output layer weights based on the derivative of the sigmoid function and target values.
    iv. Propagates gradients backward through hidden layers.
    v. Updates weights for all neurons.

c. Training Loop:
    i. Iterates through training data.
    ii. Performs forward and backward propagation.
    iii. Updates weights based on calculated gradients.

4. Usage

a. Input Data:
    i. Can be provided manually through the CreateTrainingData method or read from CSV files using CreateTrainingDataFromCSV.

b. Training:
    i. Training iterations and learning rate are specified in the train method.

c. Output:
    i. Prints the neural network's output before and after training.

5. Points To Note

a. Neural Network Configuration:
    i. Corrects an issue related to the number of weights in hidden layers.
    ii. Noted the need for 2 weights per neuron in the first hidden layer.

b. Data Handling:
    i. Reads input and output data from CSV files using OpenCSV.

c. Code Structure:
    i. Utilizes a modular structure with distinct classes for layers, neurons, and utility functions.

d. Activation Functions:
    i. Implements both sigmoid and ReLU activation functions.

e. Training Process:
    i. Implements a standard backpropagation algorithm with gradient descent.

f. Random Initialization:
    i. Neuron weights are initialized randomly within a specified range.

6. Conclusion

The provided code serves as a foundation for a basic neural network framework. It includes functionalities for forward and backward propagation, random weight initialization, and CSV data handling. Further enhancements may involve extending activation functions, optimizing training parameters, and improving modularity.