Two-layer Neural Networks

Pseudocode

This pseudocode outlines the steps for a two-layer neural network with backpropagation.

Given:

- **Network Type:** A two-layer neural network.
- **Domain:** A function f:X→Y is to be approximated within the domain [−1,1].
- **Sample Data Points:**
  - **Input Data (X):** A 1x21 array of values from -1.0 to 1.0.
  - **Target Data (Y):** A 1x21 array of corresponding output values.
- **Required Algorithm:** A custom backpropagation algorithm (no external packages).
- **Plotting Requirements:**
  - A plot of training error vs. epoch number.
  - A plot of the actual function f(x) vs. the neural network output at 10, 100, 200, 400, and 1000 epochs.
- **Activation Functions for Assessment:**
  - tanh
  - logsig
  - tansig
  - radialbasis
  - relu

1. Data Preparation

- Define input data X as a 1x21 array of values from -1.0 to 1.0. This array is then **reshaped into a 21x1 matrix**, where each row represents a single data point.
- Define target data Y as a 1x21 array of corresponding output values, which is also reshaped into a **21x1 matrix**.
- Determine the number of samples, m, from the shape of the input data X.

2. Network Initialization

- Define a TwoLayerNN class.

- The constructor __init__ takes input_size, hidden_size, output_size, and optional activation functions.
- Set the **learning rate** (lr) to 0.01.
- Initialize weights W1 and W2 and biases b1 and b2 using **He/Kaiming initialization** rules for the tanh activation function.
    - W1: (input_size, hidden_size) matrix with random values scaled by 2/input_size
    - b1: (1, hidden_size) matrix of zeros.
    - W2: (hidden_size, output_size) matrix with random values scaled by 2/hidden_size .
    - b2: (1, output_size) matrix of zeros.
- Store intermediate variables (Z1, A1, Z2) for use in backpropagation.

3. Training Loop

- Define a train_network function that takes the network object nn, input X, target Y, and number of epochs.
- Iterate for a specified number of epochs.

Inside the Epoch Loop:

1. **Forward Pass**:
    a. Call the forward_pass method with input X.
    b. **Hidden Layer**: Calculate the weighted sum of inputs and bias (Z1 = X @ W1 + b1).
    c. Apply the tanh activation function (A1 = tanh(Z1)).
    d. **Output Layer**: Calculate the weighted sum of hidden layer outputs and bias (Z2 = A1 @ W2 + b2).
    e. Apply the linear activation function (A2 = linear(Z2)).
    f. Return the final output A2.
2. **Loss Calculation**:
    a. Calculate the **Mean Squared Error (MSE) loss** using the formula 0.5×(Y−A2)2.
    b. Store the loss value for plotting.
3. **Backpropagation**:
    a. Call the backpropagation method with inputs X, Y, and the network output A2.
    b. Calculate gradients for all weights and biases using the chain rule.

c. **Output Layer Gradients**:
   i. Compute the error term Delta2 for the output layer. The derivative of the linear activation is 1, so Delta2 = (A2 - Y).
   ii. Calculate the gradient of the loss with respect to W2 (dW2 = A1.T @ Delta2 / m).
   iii. Calculate the gradient of the loss with respect to b2 (db2 = sum(Delta2, axis=0) / m).

d. **Hidden Layer Gradients**:
   i. Compute the error term Delta1 for the hidden layer (Delta1 = (Delta2 @ W2.T) * tanh_prime(Z1)).
   ii. Calculate the gradient of the loss with respect to W1 (dW1 = X.T @ Delta1 / m).
   iii. Calculate the gradient of the loss with respect to b1 (db1 = sum(Delta1, axis=0) / m).

4. **Weight Update**:
   a. Call the update_weights method.
   b. Update weights and biases using **Gradient Descent**:
      i. W1 = W1 - lr * dW1
      ii. b1 = b1 - lr * db1
      iii. W2 = W2 - lr * dW2
      iv. b2 = b2 - lr * db2

4. Visualization

- Plot the Mean Squared Error (MSE) loss against the number of epochs to visualize the training progress.
- Plot the original function and the network's approximation at different epochs to show how the model learns the underlying relationship.