Network Class API Documentation

1 Overview

The `Network` class is a simple implementation of a feedforward neural network for supervised learning. It supports backpropagation and mini-batch gradient descent to train the network using customizable activation functions (sigmoid or linear). This class provides functionality for training, evaluation, and logging of the training process.

2 Class Methods and Attributes

2.1 Class Attributes

- `weights`: A list of weight matrices representing the parameters of the neural network. Each weight matrix is randomly initialized during the construction of the network.
- `activation_function`: The activation function to be used by the network. It can be either `sigmoid` or `linear`.
- `activation_function_prime`: The derivative of the activation function, which is used during the backpropagation process.

2.2 Constructor: `__init__`

```
__init__(shape, activation_function='sigmoid')
```

- **Parameters**:
  - `shape` (list): A list defining the number of neurons in each layer of the network.
  - `activation_function` (str, optional): The activation function used by the network. Defaults to `'sigmoid'`. Can also be `'linear'`.
- **Description**: Initializes the network's weights and sets the activation function based on the specified type. Random values are assigned to the weight matrices, and a bias term is appended to each input.

2.3 Training: `train`

```
train(X, Y, training_rate=0.01, batch_size=None, epochs=1, log=False, log_name=None)
```

- **Parameters**:
  - `X` (ndarray): The input data matrix (samples × features).
  - `Y` (ndarray): The target output matrix.
  - `training_rate` (float, optional): The learning rate for the gradient descent algorithm. Defaults to 0.01.
  - `batch_size` (int, optional): The number of samples per batch. If not provided, defaults to the total number of samples.
  - `epochs` (int, optional): Number of training iterations over the entire dataset. Defaults to 1.
  - `log` (bool, optional): Whether to log the training progress into a file. Defaults to `False`.
  - `log_name` (str, optional): The name of the log file where progress is saved. Required if `log` is set to `True`.
- **Description**: Trains the network using mini-batch gradient descent over the specified number of epochs. If logging is enabled, the progress and weights are saved to a log file.

2.4 Evaluation: `evaluate`

```
evaluate(X)
```

- **Parameters**:
  - `X` (ndarray): The input data to be evaluated (samples × features).
- **Returns**:
  - `output` (ndarray): The network's output after applying the forward pass.
- **Description**: Feeds the input data through the network, returning the final output predictions.

2.5 Backpropagation: `backpropagation`

```
backpropagation(X, Y)
```

- **Parameters**:
  - `X` (ndarray): The input data for the current batch.
  - `Y` (ndarray): The corresponding target output for the current batch.

- **Returns**:
  - `gradients` (list): The gradients of the cost function with respect to the weights for each layer.
  - `total_error` (float): The total error for the batch.
- **Description**: Performs the backpropagation algorithm to compute the gradients for the current batch and returns the error.

2.6 Feedforward: `feedforward`

```
feedforward(X)
```

- **Parameters**:
  - `X` (ndarray): The input data.
- **Returns**:
  - `zs` (list): The weighted sums for each layer.
  - `activations` (list): The activations for each layer.
- **Description**: Propagates the input data through the network, storing the activations and the weighted sums for each layer.

2.7 Gradient Descent: `gradient_descent`

```
gradient_descent(gradients, training_rate, batch_size)
```

- **Parameters**:
  - `gradients` (list): The gradients of the cost function with respect to the weights.
  - `training_rate` (float): The learning rate for the gradient descent step.
  - `batch_size` (int): The size of the current mini-batch.
- **Returns**:
  - `new_weights` (list): Updated weights after applying gradient descent.
- **Description**: Updates the network's weights using gradient descent and the calculated gradients.

2.8 Helper Functions

2.8.1 Cost: `cost`

```
cost(Y, output)
```

- **Parameters**:
  - `Y` (ndarray): The target output.
  - `output` (ndarray): The predicted output.
- **Returns**:
  - `cost` (float): The cost between the target and the predicted output.

2.8.2 Cost Derivative: `cost_derivative`

```
cost_derivative(Y, output)
```

- **Description**: Computes the derivative of the cost function with respect to the output.

2.8.3 Activation Functions

`sigmoid(z)` and `sigmoid_prime(z)` implement the sigmoid activation function and its derivative.

`linear(z)` and `linear_prime(z)` implement the linear activation function and its derivative.

2.8.4 Batch Creation: `create_batches`

```
create_batches(X, Y, batch_size)
```

- **Parameters**:
  - `X` (ndarray): Input data.
  - `Y` (ndarray): Target output.
  - `batch_size` (int): The number of samples in each batch.
- **Returns**:
  - `batches_X`, `batches_Y`: Lists of mini-batches for input and target data.