Deep Learning Assignment 1 : Neural Network

Student Name: Gaurav Khatri

This document provides a general code documentation of the Deep Neural Network design codebase. The entire architecture is divided into six main modules and a single notebook file. A brief explanation of each of the classes is as follows:

Module 1 : DataLoader

The DataLoader class is a useful utility for loading and preprocessing the MNIST dataset and generating batches of training data during the training process.

Module 2 : Activations

This module defines activation classes, RELU and Sigmoid, both inheriting from a base Layer class, with the options to add new activations in the future.

Both of the class implements the forward and backward pass of the activation functions. For example, In ReLU ,the forward pass simply applies the max function to the input data with 0 as the second argument. The backward pass calculates the gradient of the output with respect to the input data, which is 1 for input data greater than 0 and 0 for input data less than or equal to 0.

Module 3 : Loss Functions

This module code loss functions for use in neural networks: CrossEntropy and HingeLoss. Each loss function is a subclass of a base class called Layer, which provides some common functionality.

CrossEntropy is a loss function commonly used for multi-class classification problems, and it includes methods for computing the log-softmax and stable softmax of the predicted values, as well as the actual loss and gradient functions.

HingeLoss is a loss function commonly used for binary classification problems, and it includes methods for computing the actual loss and gradient functions.

Since both the loss functions and activation functions derive from same base functions, we can simply add new functions in the future without the need to update the remainder of the codebase.

Module 4 : Layers

The module defines an abstract base class *"Layer"* and a derived class *"Dense"* for implementing Multilayer Perceptron (MLP) Layers. The Layer class defines the basic structure of the layer, while the Dense class defines the structure and functionality of a dense layer in an MLP. The Dense layer implements forward and backward propagation of inputs to compute the output of the layer and propagate gradients back through the layer, updating the layer's weights and biases using gradient descent. The layer can be initialized with random or zero weights using the weight initialization parameter. The learning rate parameter controls the step size taken during gradient descent.

Module 5 : Sequential

This module defines a class called Sequential for creating a sequential neural network. The class contains several methods for adding layers to the network, printing the architecture, constructing the network based on input parameters, and training the network using a specified loss function.

The Sequential class has an attribute called network which is initially an empty list that is populated with the layers of the network using the add and add-list methods.

The "*construct_network*" method is used to define the architecture of the network by creating instances of Dense layers with the specified input and output units, number of hidden layers and hidden units, and activation function.

The *"forward"* method computes the forward pass of the input through the layers of the network and returns the list of activations.

The train method is used for training the network using the specified loss function. It performs forward and backward propagation for each layer in the network, and updates the weights and biases of each layer using the gradient of the loss function.

Module 6 : Helpers

The module contains four functions that implement a neural network training and testing process.
The *"process_data"* function downloads data, splits it into training and testing sets, and creates one-hot encodings of categorical outputs.
The *"get_accuracy"* function calculates the accuracy of predicted and true labels.
The *"main"* function trains the neural network, computes the loss and accuracy, and saves the results.
Finally, the *"_plot"* function visualizes the training and testing accuracy and loss curves.

Further information about usage of these modules to train the neural network is reprsented in the Jupyter notebook.