Task 1A – Build a Fully Connected 2-Layer Neural Network to Classify Digits

NOTE: Please go through Extra_Code_Instructions.pdf for the Coding Instructions.

This document explains the problem statement to be solved in the sub-task **Task 1A**, the organization of sub-folders, how to use the folder content and programming instructions.

Problem Statement:

To classify digits with a 2-Layer Neural Network using MNIST dataset.

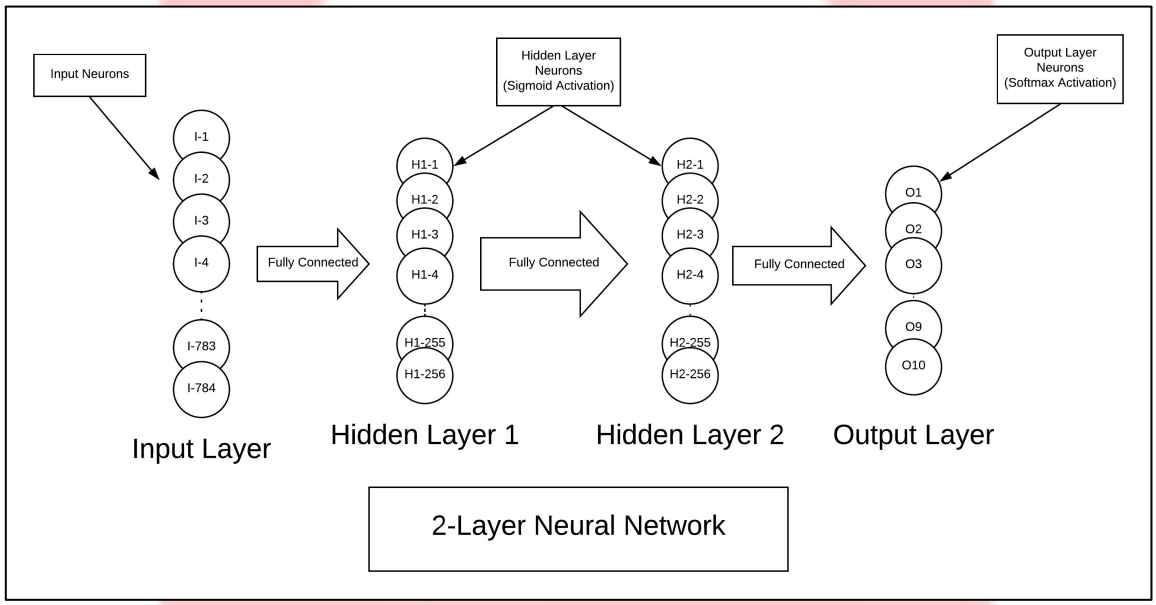

Figure 1: Neural Network Architecture

Objective:

The task requires you to code, from scratch, all parts of a two layer neural network to train and predict digits given images. We will use architecture described in Figure 1. We will make use of **PyTorch's Tensor API only** to implement all the core neural network operations and algorithms.

Where to write the code ?

In this task, you will write **Python programs in files** present inside folder named **Code**. We have made a general scaffolding for you. You are allowed to **add more functions to it, add more modules or other files if you need to**, but you should **strictly follow and write code on top of minimum structure** we have set up for you.

The functions, modules and classes should complete their objectives. Objectives will be to return a specific result in a specific format. Please, conform yourself to the format very strictly. If you fail to do this your credits for the task might not be taken into account.

We assume that, by now, you have understood the steps performed when building a machine learning solution. The steps will be the same here. Only the model architecture (here, a neural network) and its training will differ.

We have divided the entire Python application into modules that take care of different parts like ***nnet package, test package***, etc. You will use a composition of all these modules to solve the problem in ***main.py*** and present it in ***Task_1A.ipynb***.

The purpose behind doing this is to have an API for the code we write so we can reuse it. Checkout ***main.py*** for hints on how these modules will be used in conjunction with each other. Checkout the folder structure now.

Code Structure:

Now, that you are somewhat aware of the modules of the application, we will brief them here.

- ***nnet package***

  This package contains code related to different parts of neural network. You will work with **4 files** in this folder which contain code to:

  1. ***model.py*** : instantiate a 2-layer neural network. It supports a simple but expressive API to use the instantiated model.
  2. ***loss.py*** : calculate cross entropy loss and its gradient
  3. ***activation.py*** : contains functions to calculate sigmoid and softmax activation. Also calculates gradients of activation functions.
  4. ***optimizer.py*** : update parameters of neural network.

- ***test package***

  This package contains modules that test each of the modules from ***nnet package***. By default, they are checking for dimensions and type of outputs. You can write more robust cases for validating your code.

- ***main.py***

  This is the actual entry point to your application. It will use parts from above after loading and pre-processing data, and then train a model on the data.

- ***Task_1A.ipynb***

  This notebook should present the work you did in "***.py***" files in the **Code** folder. Instructions about what to do in this file are mentioned inside it.

Dataset:

The dataset we will use is MNIST. It is available for download from ***torchvision*** module and will be dowloaded only at the first time you run the code.

Resources:

Whatever resources you will need for completing this task are present in the **Task 1A *Resources*** folder. The resources can be links to best tutorials available on the web like videos, notes, etc. If required, we will also provide our own notes and videos.

Please note that we assume that you have completed all tasks before this task. So we may not include resources from previous tasks again. If you think something that you cannot complete the task without, is not present as a resource in this folder, checkout previous tasks resources. We will mark wherever required but if we don't please do as said above.

What piece of code to change and what not to change ?

Generally, all the code that is there should be left as it is. At some very important places, we will have mentioned it **explicitly**.

You should add code at all the seemingly incomplete locations and use your conscience to check if the function is doing what is intended of it. You can verify if your output conforms to the basic requirements using test modules but don't rely on the default tests too much to know if your implementation is right.

You are allowed to create more modules and packages to structure your code better but **the functions, modules, etc. that we have documented should return exactly what is asked of them. You cannot change names of these files or functions or modules, in some cases even variables.**

What libraries can we use or can not use ? (Important)

To implement all the algorithms we will use only **PyTorch Tensor API**. Tensor API will look like NumPy but the difference is that it can also execute on GPU. You can use Pandas, Matplotlib and minimal NumPy (only for operations that are not supported by PyTorch's Tensor API). You cannot use higher level PyTorch APIs (like **nn**) in ***nnet package*** but you can use it to test your implementation against it in ***test package***.

Where to go next from here ?

The most important thing you need to know to complete the task is the **theory of neural networks**. Checkout the ***Resources*** folder.

Try to understand overall neural networks theory and then check out each part and it's math in detail. Once you think you understand enough of something small, not necessarily the entire neural network theory but a small part of it in detail, you can then go and implement that part in code.

Use resources as reference or take notes when doing this. One by one, complete each part and in no time you'll be done with complete task.

…Best Wishes ! …