Lab Assignment 1

1 Overview

Welcome to the first lab assignment for Neural Networks and Deep Learning! Please be sure to **read this document in its entirety**. In this assignment, you will apply your knowledge of the back-propagation algorithm studied in problem set 2 to build a multi-layer perceptron (MLP) training framework using the MNIST dataset. Then, you will use this powerful tool to create and design an experiment of your choosing. Finally, you will practice your technical writing by submitting a report to convey your experimental findings. This assignment is worth a total of **50 points**, split into two parts worth **40** and **10** points, respectively. These parts are: (1) building the MLP training framework, and (2) designing the experiment and crafting the report. An extra credit opportunity (**5 points**) is provided if you are interested in extending the framework. Background on the MLP model is provided in the next section.

2 Background

2.1 The MLP Model

For this assignment, we are only concerned with multi-layer perceptron models with layers of the following form:

$$L_i(x_{i-1}) = f_i(x_{i-1}\mathbf{W}_i + \mathbf{b}_i) \tag{1}$$

Here, L_i denotes the parameterized function learned by the i^{th} layer. $f_i(\circ)$ is the non-linear activation function used by the the i^{th} layer, and $\mathbf{W}$, $\mathbf{b}$ are the learned weights and biases. The full model (M) can be formulated as follows:

$$M(x) = L_n(L_{n-1}...(...L_2(L_1(x)))) \tag{2}$$

For our purposes, $x \in \mathbb{R}^{b\times d}$ denotes a batch (of size b) d-dimensional input features. Please note that we will follow the convention that the batch size determines the first axis of the input tensor. We denote by d_i the dimension of the *hidden* features from layer i, i.e.

$$L_i(x_{i-1}) : \mathbb{R}^{b\times d_{i-1}} \to \mathbb{R}^{b\times d_i} \tag{3}$$

The MLP *architecture* is primarily defined by the choice of d_i, i.e., how many hidden neurons are used for each layer. Please note the following:

$$\begin{aligned} \mathbf{W}_i &\in \mathbb{R}^{d_{i-1}\times d_i} \\ \mathbf{b} &\in \mathbb{R}^{1\times d_i} \end{aligned} \tag{4}$$

You are encouraged to verify that equation (1) makes sense given these parameter shapes. Note the bias is added to each of the b samples, i.e. equation (1) is shorthand for:

$$L_i(x_{i-1}) = f_i(x_{i-1}\mathbf{W}_i + \mathbf{1}\mathbf{b}_i) \tag{5}$$

where $\mathbf{1} \in \mathbb{R}^{b\times 1}$ denotes a column vector of all 1s. Also note that the non-linearity f is applied element-wise to the input, i.e:

$$L(x)[i,j] = f(P[i,j]) \ , \forall i,j \tag{6}$$

where $P = x\mathbf{W} + \mathbf{b}$ denotes the pre-activation values.

2.2 The MNIST Dataset

MNIST is a popular dataset used in the machine learning and deep learning communities. It consists of 60000 training images and 10000 test images composed of black-and-white handwritten digits from $0-9$. The images are of the shape 28×28, however, we will work exclusively with *flattened* versions of shape $28^2 = 784$. Flattened means that we reinterpret the 2-D images as 1-D vectors instead. Please note that this means d_0, the dimensionality of the input to the MLP model, **must always be** 784. Similarly, d_n, the dimensionality of the output, **must always be** 10, for the 10 classes. A .zip file containing the MNIST dataset is supplied for you on the Canvas assignment page.

3 Building the MLP Framework: 40 Points

3.1 Overview and Permitted Functions

The goal of the MLP framework is to allow you to easily experiment with different architectures, activation functions, and hyper-parameters. A starter code-base has been provided for you on the Canvas assignment page. This code-base is also available in Jupyter notebook form, if you prefer to work in that environment. Please make a copy of the notebook and save it to your personal drive in order to edit the code. Notebooks and regular python scripts are both acceptable for this assignment; use whichever you feel more comfortable with. Most of the structure of the program is already in place; you only need complete portions of some of the functions. Everything should be written in Python, and you are allowed to leverage some utility functions from the PyTorch library. However, you are **NOT** permitted to use any of the following PyTorch modules/functions:

1. Linear() or the functional version thereof
2. loss.backward()
3. The MLP module
4. CrossEntropyLoss()
5. Any pre-configured linear layer/MLP module

In other words, you must implement the MLP model forward/backward pass yourself. Any submission which relies on built-in MLP tools will receive **zero credit**. You are freely permitted to use any tensor operation functions such as matmul(), where(), mean(), etc. Note that we will use a python list of the form $[d_0, d_1, ..., d_n]$ to represent how many neurons are used for each layer. This list can be accessed from the main MLP class used in the code-base. As an example, to define an MLP with 2 hidden layers of size 1024, we would use the following:

$$[784, 1024, 1024, 10]$$

3.2 Loading the Dataset

First, download the .zip file containing MNIST and extract/un-compress the files. Code to read in the data as tensors is already provided for you; all you need to do is update the "base_path" variable to point to wherever you downloaded the files. Please note if

you choose to write your code on Google Colab, refer to the .ipynb file on the Canvas assignments for instructions on how to upload the data to Google drive for use with that environment. After you have loaded the dataset with the "mnist.load_data()" and "normalize_mnist()" functions, you will have 4 tensors containing the train/test inputs/labels.

3.3 Initializing the MLP Model

You will need to complete the "initialize()" function. This function should populate the "self.weights" and "self.biases" class variables with tensors of the appropriate shape (equation 4). Initialize all bias terms to zero, and all weight terms according to the following:

$$\mathbf{W}[i,j] \sim \mathcal{U}\left(-\sqrt{\frac{6}{d_{i-1}+d_i}}, \sqrt{\frac{6}{d_{i-1}+d_i}}\right) \ , \forall i,j \tag{7}$$

where $\mathcal{U}(x,y)$ denotes the uniform distribution with across the range $[x,y]$. This is known as Xavier Uniform initialization, which facilitates numerical stability in the MLP layers.

3.4 Implementing ReLU and Leaky ReLU

You will need to complete the code for the (Leaky) ReLU activation functions. For context:

$$\begin{aligned} ReLU(x) &= max(x,0) \\ LeakyReLU(x) &= max(x,0.1x) \end{aligned} \tag{8}$$

The code-base provides classes for each of these activation functions; you just need to fill out the "forward()" and "backward()" functions.

3.5 Implementing the Forward Pass

The "forward()" function takes in an input $x \in \mathbb{R}^{b \times d_0}$ and passes it through all MLP layers via equations 1 and 2. Note that PyTorch will automatically "broadcast" the addition of the bias term, you do **NOT** need to explicitly implement equation (5). Also note that the MLP class contains an "activation_function" variable which can be set to either ReLU or Leaky ReLU. You will need to complete the "forward()" function for these activations in order to use them in the framework. After the final layer, use the softmax function to convert the output into a probability distribution. Recall that softmax is defined as:

$$softmax(x)[i] = \frac{e^{x[i]}}{\sum_{i=1}^{d} e^{x[i]}} \tag{9}$$

In other words, softmax normalizes the input by the sum after exponentiation by e. To summarize, the forward pass should send the input through all MLP layers using the desired activation function, then softmax should be applied to the final output. **IMPORTANT NOTE:** You *must* store the intermediate "hidden" features computed at each layer in the "self.features" class variable. These will be necessary in order to perform back-propagation later.

3.6 Computing the CE Loss

We will use the cross-entropy (CE) loss for the framework. After the softmax-ed outputs have been obtained from the forward pass, CE can be computed as follows:

$$CE(Y, P) = -\sum_{c=1}^{C} Y[i] log(P[i]) \tag{10}$$

Here, C denotes the total number of classes, Y is a one-hot encoded ground truth vector, and P is the predicted class probabilities after softmax.

3.7 Implementing the Backward Pass

After the forward pass and resulting loss have been computed, you will need to back-propagate the error through the MLP model and update all the weights and biases. The starting code-base accepts a "delta" argument as input to the "backward()" function. You should supply the gradient **after** back-propagation through softmax as this value. This means that in the "backward()" function, you only need to back-prop through equations 1 and 2. To get you started, recall the following:

$$\frac{dCE}{dO} = \frac{dCE}{dP}\frac{dP}{dO} = Y - P \tag{11}$$

Here, Y is the one-hot encoded ground truth, P is the predicted probabilities after softmax, and O is the output of the MLP model *before* softmax is applied. The values O are often called *logits*. You should pass the value computed via equation (11) to the "backward()" function. From there, derive and implement the back-prop equations for the MLP model in order to obtain the following:

$$\begin{gathered} \frac{dCE}{d\mathbf{W}_i} \\ \frac{dCE}{d\mathbf{b}_i} \end{gathered} \tag{12}$$

for all layers. Then, update the model's parameters according to the rule:

$$\begin{aligned} \mathbf{W}_i &\leftarrow \mathbf{W}_i - \eta\frac{dCE}{d\mathbf{W}_i} \\ \mathbf{b}_i &\leftarrow \mathbf{b}_i - \eta\frac{dCE}{d\mathbf{b}_i} \end{aligned} \tag{13}$$

for all layers. The learning rate η is stored in the MLP class as the "learning_rate" variable. Note that when using a batch size greater than 1, you will need to compute the average gradient over the batch before updating the parameters with equation (13).

3.8 Putting it all Together

The core structure of the "TrainMLP()" function has been provided for you. All you need to do in order to start using your framework is to call the "forward" and "backward" functions in the correct places. Additionally, you should compute the train and test loss. Finally, the "main" function provides a sample implementation of a simple MLP

model using the framework. Please verify that your solution is correct by training this architecture as-is, before making any adjustments for your experiments. This default set-up uses an architecture of $[784, 256, 10]$ with ReLU activation, a learning rate of $1e-6$, and a batch size of 512. When trained for 25 epochs, this should result in approximately 85% test accuracy.

4 Using the MLP Framework: 10 Points

Congratulations! You've just built a very powerful tool, and now it's time for the fun part! Your task for this portion is to design an experiment to investigate the importance of MLP architectural details. The experiment should explore the importance of *model depth*, i.e. the number of layers, and *model width*, i.e. the number of hidden neurons. You should use test accuracy as the primary measure of success for the experiment, although you're more than welcome to include train/test loss in your results as well.

The specifics of this experiment are left up to you! The only requirement is that 5 or more distinct architectures are explored, with a decent variety of number of layers and hidden layer sizes. Discuss in your report what you observe. Do more layers help? What about wider layers? Can you get away with very "narrow" layers if enough layers are used? These are just some possible questions you can investigate. At least 2 *trends* should be discussed in your analysis, with supporting evidence to back them up.

5 Deliverables

5.1 Part 1

Submit your code for: (1) the "forward" function, (2) the "backward" function, and (3) the "initialize" function. Copy-paste these three functions into a single .py file for submission. Additionally, submit a screenshot demonstrating that your framework achieves $\sim 85\%$ test accuracy when using the default configuration (see section 3.8). The screenshot should include the "main" function and the corresponding test accuracy. Do not worry if your code does not result in exactly 85%, there will always be some run-to-run variance due to random initialization. You should get within $\pm 1\%$ however.

5.2 Part 2

Submit an approximately 1 page report (can be longer if desired) discussing your experimental design, results, and analysis for section 4. This report should be well formatted, clearly understandable, and convey what you found. Details on each section of the report are discussed in the next sections.

5.2.1 Experimental Design

The first section of your report should discuss *what* your experiment aims to study (e.g. are deeper MLPs better?), *how* it will be conducted (which variables will be changed, which left fixed as controls, etc...), and *what* you expect to find (hypothesized outcome). You **MUST** report all configuration details (including hyper-parameters!) used in your study. When writing this section, ask yourself "could someone else reproduce this experiment solely based on the report?" If the answer is no, then you need to add detail.

5.2.2 Results

The second section of your report should provide the results of your experiments. This can be accomplished with plots, tables, charts, etc. Your figures **MUST** be clearly formatted, well annotated, and understandable by anyone. Remember to include labels for axes, titles/captions for figures, legends, etc. There should be *zero* ambiguity about what data your results are trying to convey.

5.2.3 Analysis

The third and final section of your report should discuss what you learned from your experiments. This should be more than a superficial reiteration of what can be observed in your results. Do **NOT** simply state what the results are, rather discuss *what the results say*. What trends did you observe from the experiments? Was your original hypothesis supported or refuted? As an example, consider the table of fictional results below:

Model	Activation	Accuracy
Model 1	Leaky	50%
Model 1	ReLU	45%
Model 2	Leaky	55%
Model 2	ReLU	57%

It would be insufficient to say "Model 1 achieves 50%, 45%, while Model 2 achieves 55%, 57%." A better analysis would be: "Model 2 always outperforms Model 1, but exhibits different behavior with respect to the activation function used. Model 1 performs better with Leaky ReLU whereas Model 2 performs better with ReLU."

6 Extra Credit: 5 Points

The goal of the extra credit is to add extra functionality to your MLP framework. Choose any 2 items from the below list to implement, then submit screenshots of your code.

1. Support for the *tanh* and sigmoid activation functions.
2. Support for L1 Regularization.
3. Support for L2 Regularization.

After implementing 2 of the above items, provide a short (1-2 paragraph) discussion on how these extra features relate to model performance. Choose one architecture from your experiment in part 2 and compare the results before and after adding the extra features to your framework. Do the additions improve performance? Reduce it? This does not need to be a full report, a brief discussion on the key take-aways is sufficient.

Collaboration versus Academic Misconduct: Collaboration with other students (or AI) is permitted, but the work you submit must be your own. Copying/plagiarizing work from another student (or AI) is not permitted and is considered academic misconduct. For more information about University of Colorado Boulder's Honor Code and academic misconduct, please visit the course syllabus.