Chapter 2

Multi-Layer Perceptron

Professor:

José Oramas Mogrovejo

Teacher Assistants:

Hamed Behzadi Khormouji, Arian Sabaghi

Artificial Neural Networks

February 25, 2024

1 Introduction

This first practical session introduces a multi-layer perceptron in practice. This network include some layers namely, input, hidden, and output layer. This session covers practical implementation of some aspects learned in theory lectures, namely, **Algebraic [Matrix] formulation, ReLU activation function, loss functions, Multiclass classification, Gradient Descent, and Back-propagation Algorithm.**

1.1 Learning Outcome

a) You will be able to utilize Algebraic [Matrix] Formulation to create *Forward* and *Backward* operations in a shallow neural network.

b) You will learn in a practical manner implementation of your designed formulations.

c) You will experience designing a training algorithm for a multi-class classification.

d) You will obtain a deep intuition of gradients' effect in training neural networks.

2 Environment

In this session, you are free to choose the environment that is more convenient for you. This could be either Google-Colab, IDLab's GPULab, or your local machine.

3 Dataset and Code

You will find code for this exercise in the BlackBoard platform located at the address: *Artificial Neural Networks/Coding Sessions/Session-2 MLP*. The *Python* file name is *two_layer_network_st*. In this session, we create a random dataset of numbers.

4 Training A Neural Network

4.1 Architecture

Figure 1 illustrates the architecture of a network with one hidden layer. In this architecture, input size and output sizes are 1000 and 10, respectively. The hidden layer has 100 neurons. Each forward link between neurons shows a weight between those neurons. These weights are the parameters that need to be trained.

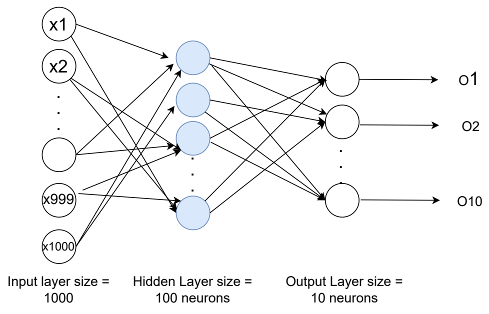
x1
x2
x999
x1000
o1
O2
O10
Input layer size = 1000
Hidden Layer size = 100 neurons
Output Layer size = 10 neurons

Figure 1: Architecture of a network with one hidden layer.

4.2 Problem 1 (15 Minutes)

According to the designed architecture:

a) Write in a paper math formulas for *Forward* pass operation.

b) Write loss function and *Backward* pass math formula. Since we use *ReLu* activation function, what is the first order derivative of this function?

c) Implement the written formulas in the provided script file. This should be done in the *for* loop implemented in the file. The comments starting with # *Problem 1* and each sub-comment written in the code will help you where these formulas should be implemented. When you run the code, you are presented by a figure which shows the distribution of data for three classes in the space.

4.3 Problem 2 (15 Minutes)

In this section you are asked to:

a) Write math formula for updating the parameters of network based on *Gradient Descent* algorithm.

b) In the next step, implement the formula in the script file. The comment starting with # *Problem 2* will guide you to the line you should implement desired code.

c) Now, change the value of the seed in the code and run the program. Is there any change in the loss curve or the latest value of loss? How do you justify this?

4.4 Problem 3 (5 Minutes)

In this section, you are asked to answer questions about the intuition of calculating gradient in *BackPropagation* operation.

a) What does sign and magnitude of the gradient represent in the learning problem?

b) Why do we subtract the proportion of the gradient value from each parameter? What happens if we sum the value of parameters with the gradient magnitude?

5 Constructing a functional Neural Network

This section provides a modified example from *Pytorch* tutorial. This example code trains a two-layer network implemented in a functional way using images with size 3x32x32 from the image dataset CIFAR-10. The goals of this section are as follows:

- Providing a foundation for further practical sessions.
- Familiarity with *Pytorch* libraries that provide flexible means for designing architectures and training algorithms.

5.1 Files

Locate the FunctionalNN folder in the downloaded contents. The comments will make you familiar with some *Pytorch* functions for designing and training deep models.

- *main.py* calls other functions and classes.
- *settings.py* defines hyperparameters of the model.
- *Dataset.py* loads the data and applies transformations on the data.
- *Model.py* implements the architecture of the model.
- *helper_functions.py* implements the training algorithm.

5.2 Problem 1 (15 Minutes)

From the file *Model.py*, increase and decrease the complexity of the model. To do so:

a) You have different options for increasing the complexity of your model such as adding more layers or increasing the number of neurons in the hidden layers. For this modification run the program and save the figure of the results at the end.

b) Opposite to the previous step, you can decrease the complexity of the initially designed model by removing layers or decreeing the number of neurons in the hidden layer. After modifying the code, run the program and save the figure of the results at the end.

c) Considering the modifications applied to the model architecture, how you can justify the obtained results?

6 Text Classification

In this section, we will see how we can process text data containing tweets so that we can classify them into offensive and not-offensive categories.

6.1 Dataset

We will use OLID [1] (Offensive Language Identification Dataset) which has 14,100 tweets in total, of which 13,240 are in the training set, and 860 are in the test set. The original dataset has three levels of labels: (A) Offensive/Not-Offensive, (B) Targeted-Insult/Untargeted, and (C) Individual/Group/Other. However, for our exercise, a cleaned version of the dataset is provided which only has level-A labels (Offensive/Not-Offensive).

6.2 Files

File information for *text_classification*:

a) *data* folder which contains two tsv files for training and testing.

b) *main.py* calls all defined functions in order to train and evaluate a text classifier.

c) *dataset.py* A series of functions should be implemented to convert our raw text data to the desired format of our network's input.

d) *model.py* implements the architecture of the model.

e) *config.py* defines the required parameters and hyperparameters for training and data loading.

f) *helper.py* implements the functions related to training and evaluation loops.

6.3 Problem 1 (15 minutes)

a) What are the differences between images and texts in terms of data type?

b) How we can feed text data to a Multilayer Perceptron?

c) What preprocessing steps we may need for raw text data?

6.4 Problem 2 (35 minutes)

Considering the discussions from Section 6.3, in this section you are asked to implement code snippets to train a Multilayer Perceptron to classify the text data.

a) Implement the four functions in *dataset.py* using hints in the code. These functions should convert raw input data into a required input format for a Multilayer Perceptron. (20 minutes)

b) What should be our input size for our Model? What are the effects of changing input dimensions? (5 minutes)

c) After completing the prior task, run the program and wait for the training phase to take place. What is your evaluation of the trained model w.r.t the achieved results? How we can improve it? (10 minutes)

Reference

[1] https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge.