Deep Learning Project

Multi Layer Perceptron

42AI contact@42ai.fr
Quentin Feuillade--Montixi quentin@42ai.fr

Summary: The goal of this project is to have a first taste of neural networks, and to put the first stones of your own neural network framework

Chapter I: Introduction

In the language of your choice, you will implement a multilayer perceptron, in order to classify handwritten digits.

I.1 A bit of history

Machine learning is a vast field, and artificial neural networks is one of the main subsets in today's world, machine learning applications.

Neural networks were invented a long time ago, but it’s only been a few years since the amount of research about it has literally exploded . This is mainly due to the exponential growth of computing power and the arrival of new clever strategies.

The perceptron was invented by Frank Rosenblatt in 1957, it is a single layer linear classifier, and also one of the first neural networks to be implemented. Unfortunately the results were not as good as expected and the idea was abandoned. A bit more than 10 years later the algorithm was improved as the multilayer perceptron, and was used once again.

Chapter II: Goals

This project aims to get you acquainted with artificial neural networks, and to make you implement the algorithms at the heart of the training process.

At the same time you are going to reacquaint yourself with the manipulation of derivatives and linear algebra as they are indispensable mathematical tools for the success of the project.

You will also put the first stones of your own neural network framework (for this project you will do all by hand, but after it, you will choose either to continue to build your framework, or to learn to use existing ones with applied cases)

Chapter III: General guidelines

- This project will only be corrected by humans. You are therefore free to organize and name your files as you wish, nevertheless respecting the constraints listed here.
- Your code will have to be modular, which means that it should be easy to append a new layer. A lot of frameworks exist in all languages, We advise you to look at how they do to organize their models.
- The choice of programming language is yours, you have no restrictions on it. (it would be wise to choose a Object-Oriented programming language 🧙‍♂️)
- In the case of a compiled language, you will need to hand out a Makefile. This Makefile must compile the project, and must contain the usual compilation rules. It should only recompile and relink the program when necessary. Dependencies should also be downloaded / installed using the Makefile if necessary.
- No libraries handling the implementation of artificial neural networks or the underlying algorithms are allowed, you must code everything from scratch, you can however use libraries to handle linear algebra, matrix algebra, data manipulation and visualization.
- The norm is not applied on this project. However, we ask you to be clear and verbose about the design of your source codes.

Chapter IV: Mandatory part

IV.1 Foreword

During correction, your work will be evaluated with an anonymized dataset that you will not have access to, in addition to the one you are provided with. To make sure everything works as intended, you have to make sure you are following all the points in the implementation part.

IV.2 Dataset

The dataset provided is a custom segmentation of a well known dataset in the field : MNIST digits. The goal is, given a 28x28 black and white image (flatten on 784 columns of pixel value), to predict the digit written on the picture.

After unpacking the datas provided in ressources, you will have a csv file named train.csv. It contains values for each pixel and the corresponding label you will have to predict.

IV.3 Implementation

You will have 2 datasets, one for the training (train.csv) and the other for the evaluation (evaluation.csv). The first one should be used to train your model, and the second one will be used to create predictions for the evaluation. The evaluation will be based on accuracy score. (the evaluation.csv is an example of how the file to evaluate your algorithm will looks like during peer correction)

You must implement a MLP model that can either take a list of layers, or the parameters to build them (look at how the existing framework works). If you find more clever ways to do so, please feel free to implement it but you will have to justify your choices, and prove the modularity of your method.

Your default model (running the program without parameters) should at least contain 2 hidden layers and it must be possible to design the Neural network easily and in a fully customizable manner.

You must at least implement sigmoid, and softmax for the final layer. The activations must be applied to each layer independently.

In order to evaluate the performance of your model in a robust way during training, you must split your dataset in two parts, one for the training, and one for the validation (must be parameterized).

For the training program, you must implement minibatch gradient descent. You will see that in this dataset, it has a tremendous influence on the training time. (think why, bonus may be given for a good explanation)

To visualize your model performance during training, you will display at each epoch the training and validation metrics. For example :

```
epoch 50/100 - loss: 0.0750 - validation_loss: 0.0406
epoch 51/100 - loss: 0.0749 - validation_loss: 0.0404
epoch 52/100 - loss: 0.0747 - validation_loss: 0.0403
```

You must at least implement an accuracy metric in addition to the loss metric.

You also have to implement a learning curve graph that will be displayed at the end or during the training. It will display the loss and accuracy of your model throughout the training.

Caution : If your learning curve stagnate or start by going up too long, there is a problem in your algorithm, and it will be graded 0 (even if your algorithm ends up learning on the long run)

In your training program, you have to implement a way to save and load models from precedent training.

Your training program must take the index of the column used as label, and can take -1 for the last column. (0 for the first column, 1 for the second, etc…)

Finally you must implement another program that takes a dataset without labels (like evaluations.csv) and your model file trained with your training program. It will then output a csv file named my_answer.csv (it must be named like this !) .

The output csv file must look like this :
(the id column is optional)

rowid	label
0	$\hat{y}_0$
1	$\hat{y}_1$
2	$\hat{y}_2$
...	...
n	$\hat{y}_n$

Hint : Look at the bonus part if you're interested in bonuses before starting your implementation. Some of the bonuses will need to be implemented at the same time as the mandatory part if you don't want to lose time redoing half your code.

IV.4 Turn-in

You must submit everything you need to train your model, test it and yield results. You can also push already trained weights if your training is long.

Chapter V: Bonus part

Here is a list of the bonuses that will be graded (you can play with other things for fun but it won't be graded) :

- Implementation those activation functions :
    - Tanh
    - Relu
    - leaky Relu
    - Elu
    - Gelu
- Implementation Dropout
- Implementation of Batchnorm
- Implementation of Xavier weight initializations

Submission and peer correction

Submit your work on your ***GiT*** repository as usual. Only the work on your repository will be graded.