Deep Learning Project

Multi Layer Perceptron V2

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

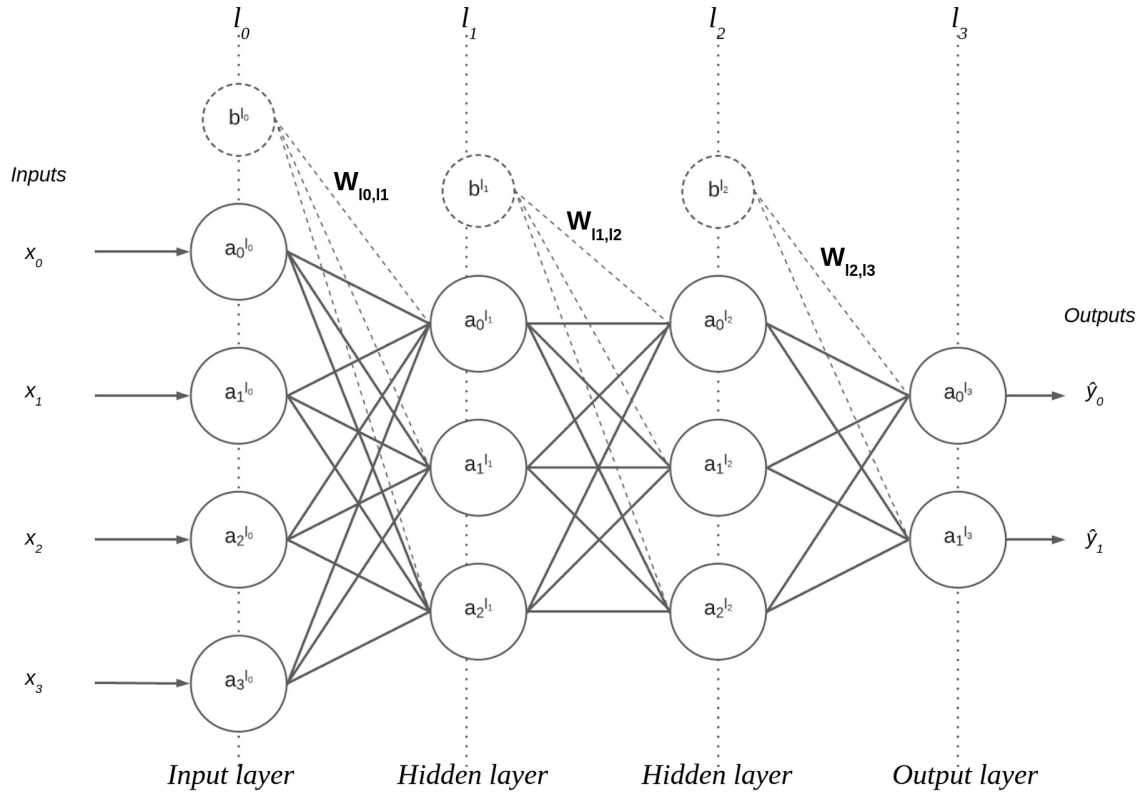
Machine learning is a vast field, and artificial neural networks is one of the main subsets in today's world, machine learning applications.

Neural networks were invented a long time ago, but it’s been just a few years that 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. In this 1948 article titled intelligent_machinery, Alan Turing introduced a type of neural networks named *B-type unorganised machine* that he considered as the simplest possible model of the nervous system.

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.

I.2 Multilayer perceptron

The multilayer perceptron is a feedforward network (meaning that the data flows from the input layer to the output layer). It is defined by the presence of one or more hidden layers, as well as an interconnection of all the neurons of one layer to the next layer.



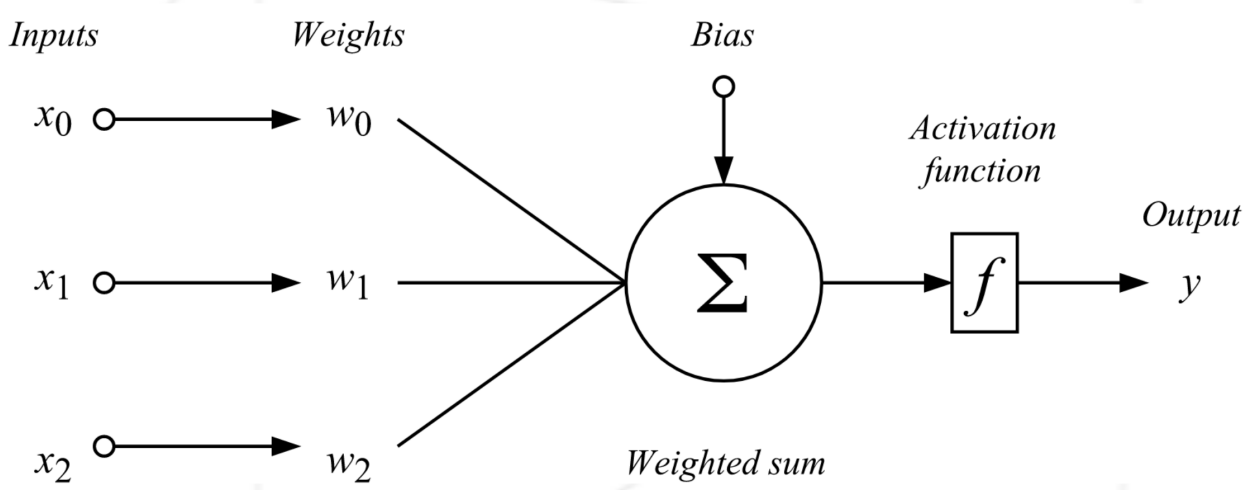
The diagram above represents a network containing 4 dense layers (also called fully connected layers). Its inputs consist of 4 neurons and its output of 2 (perfect for binary classification, but in general you have N output neurons, with N = number_of_classes). The weights connecting one layer to the next are represented by a two dimensional matrix noted $W_{lj,lj+1}$.

Matrix W_{l0l1} has the shape (3,4) for example, as it contains the weights of the connections between the layer l_0 and the layer l_1.

The bias is often represented as a special neuron which has no inputs. As a perceptron is connected to all the neurons of the following layer (the bias neurons are noted by b^{lj} on the diagram above). The bias is generally useful as it allows to “control the behavior” of a layer.

I.3 Perceptron

The perceptron is the type of neuron composing a multilayer *perceptron*. They are defined by the presence of one or more input connections, an activation function and a single output. Each connection contains a weight (also called parameter) which is learned during the training phase.



Two steps are necessary to get the output of a neuron. The first one consists in computing the weighted sum of the outputs of the previous layer with the weights of the input connections of the neuron, which gives :

$$weighted\ sum : \sum_{N-1}^{k=0} (x_k . w_k) + bias$$

The second step consists in applying an activation function on this weighted sum, the output of this function being the output of the perceptron. It can be seen as a controller which tell how much each neuron is activated (activation functions can take a lot of shapes, please refer at the bonus part to see which one you could use)

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 [or do both if you're crazy])

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

A non-negligible part of the evaluation will be based on your understanding of the training phase (also called the learning phase) and the underlying algorithms. You will be asked to explain to your corrector the notions of ***feedforward***, ***backpropagation*** and ***gradient descent***. Points will be attributed depending on the clarity of your explanations. These notions are important for the next projects of the branch and will represent a real asset if you wish to continue in this field.

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 what is the digit written.

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.

As you will see, there is an important work of data understanding before starting to implement the algorithm which will be able to classify it. A good practice would be to begin by playing with the dataset, displaying it with graphs, visualizing and manipulating its different features.

IV.3 Implementation

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. It must be possible to choose the number of neurons on each layer.

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 only an example of how the file to evaluate your algorithm will looks like during peer correction)

In order to evaluate the performance of your model in a robust way during training, you should split your dataset in two parts, one for the training, and one for the validation (the validation dataset is used to determine the accuracy of your model on unknown examples).

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 (Categorical Cross Entropy here).

You must at least implement sigmoid activation function and softmax for the final layer. (keep in mind that it will certainly not be enough to get good results). The activation must be applied to each layer independently.

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

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

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…)

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)

Finally you must implement a 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.

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) :

- 1pt by activation function:
  - Tanh
  - Relu
  - leaky Relu
  - Elu
  - Gelu
- 5pts Dropout
- 5pts Batchnorm
- 3pts Xavier weight initialization
- 2pts Why is minibatch gradient descent useful on this dataset ? !!
- 5pts student work is clean and student gave clear explanations of his work (and maybe added some functionalities) !!

!! : hidden on the pdf but used for the correction scale

Submission and peer correction

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