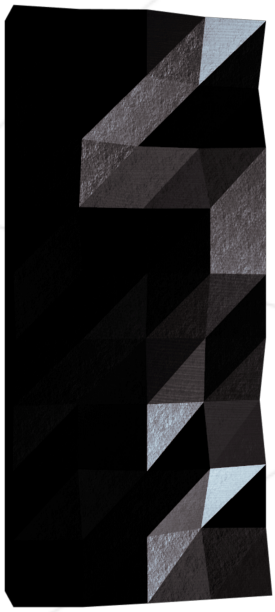


Machine Learning project

Multilayer Perceptron

Summary: *This project is an introduction to artificial neural networks, with the implementation of a multilayer perceptronls*

Version: 5

Contents

Chapter I

Introduction

In the language of your choice you are going to implement a `multilayer perceptron`, in order to predict whether a cancer is malignant or benign on a dataset of breast cancer diagnosis in the Wisconsin.

I.1 A bit of history

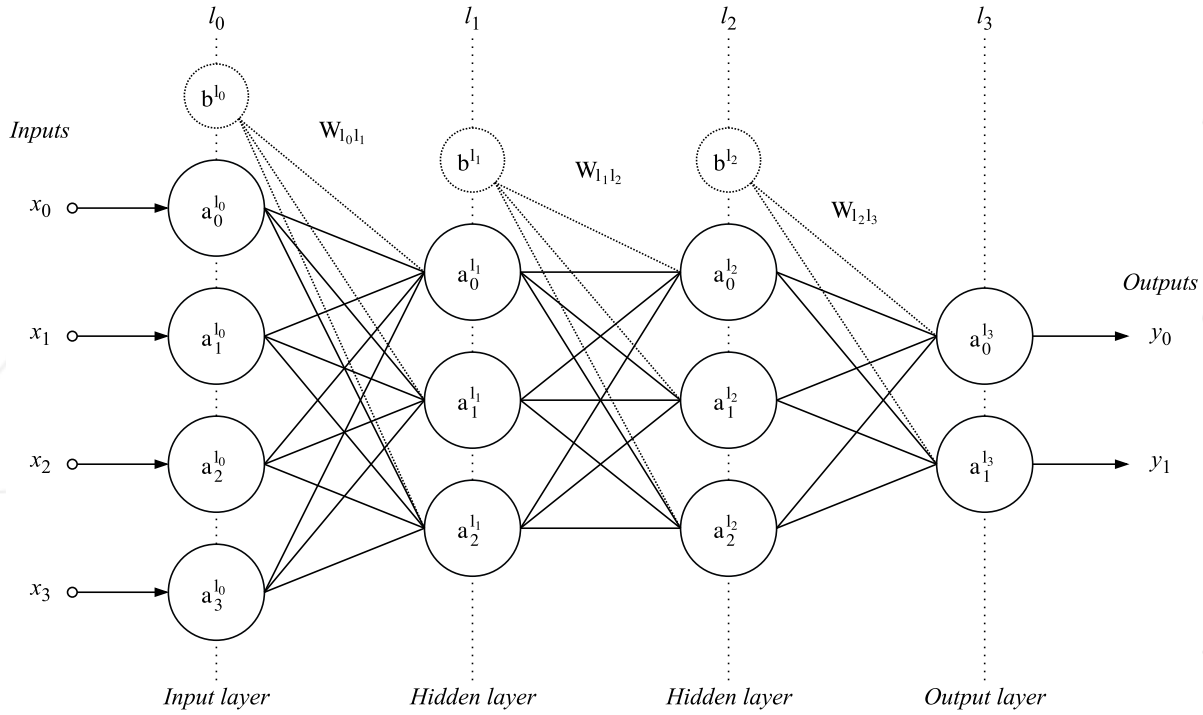
Machine learning is a vast field in which artificial neural network is only a small subset. Nevertheless we are going to tackle it since it is a really powerful tool that resurfaced a few years ago.

Conversely to what one may think, artificial neural networks have existed for a long time. In his 1948 paper '`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 network 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) 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.

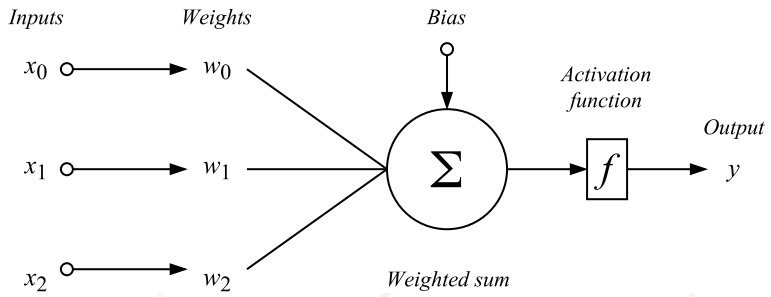


The diagram above represents a network containing 4 dense layers (also called fully connected layers). Its inputs consist of 4 neurons and its output of 2 (perfect for binary classification). The weights of one layer to the next are represented by two dimensional matrices noted $W_{l_j l_{j+1}}$. The matrix $W_{l_0 l_1}$ is of size $(3, 4)$ 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 and with an output always equal to 1. Like a perceptron it is connected to all the neurons of the following layer (the bias neurons are noted b^{l_j} 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 that the `multilayer perceptron` is composed of. 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

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

The second step consists in applying an activation function on this weighted sum, the output of this function being the output of the perceptron, and can be understood as the threshold above which the neuron is activated (activation functions can take a lot of shapes, you are free to chose whichever one you want depending on the model to train, here are some of the most frequently used ones to give you an idea : sigmoid, hyperboloid tangent, rectified linear unit).

Chapter II

Objectives

The goal of this project is to give you a first approach to artificial neural networks, and to have you implement the algorithms at the heart of the training process. At the same time you are going to have to get reacquainted with the manipulation of derivatives and linear algebra as they are indispensable mathematical tools for the success of the project.

Chapter III

General instructions

- This project will only be evaluated by Humans. You are free to organize and name your files as you desire while respecting the restrictions listed below.
- You are free to use whatever language you want, you have no restrictions on that point.
- 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 and to display the learning curves.
- It the case of a compiled language, you must submit a Makefile. This Makefile must compile the project, and must contain the usual compilation rules. It should recompile and relink the program only as necessary. The dependencies should also be downloaded/installed with the Makefile as needed.
- The norm is not applied on this project. Nevertheless, you will be asked to be clear and structured in the conception of your source code.

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.

IV.2 Dataset

The dataset is provided in the resources. It is a `csv` file of 32 columns, the column `diagnosis` being the label you want to learn given all the other features of an example, it can be either the value `M` or `B` (for malignant or benign).

The features of the dataset describe the characteristics of a cell nucleus of breast mass extracted with fine-needle aspiration. (for more detailed informations, go here).

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 by displaying it with graphs, visualizing and manipulating its different features.

You have to separate your data set into two parts yourself, one for training and one for validation.

`The data is raw and should be preprocessed before being used for the training phase.`

IV.3 Implementation

Your neural network implementation must contain at least two hidden layers by default.

The idea is to make you write a program a bit more modular (you can use a file or directly as an arguments)

For `example` by adding a hidden layer in file:

```
cat example_network_modularity.txt

network = model.createNetwork([
    layers.DenseLayer(input_shape, activation='sigmoid'),
    layers.DenseLayer(24, activation='sigmoid', weights_initializer='heUniform'),
    layers.DenseLayer(24, activation='sigmoid', weights_initializer='heUniform'),
    layers.DenseLayer(24, activation='sigmoid', weights_initializer='heUniform'),
    layers.DenseLayer(output_shape, activation='softmax', weights_initializer='heUniform')
])

model.fit(network, data_train, data_valid, loss='binaryCrossentropy', learning_rate=0.0314, batch_size=8,
     epochs=84)
```

Or for `example` with arguments:

```
python train.py --layer 24 24 24 --epochs 84 --loss binaryCrossentropy --batch_size 8 --learning_rate
     0.0314
```

You must also implement the `softmax` function on the output layer in order to obtain the output as a probabilistic distribution.

In order to evaluate the performances of your model in a robust way during training, you will 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).

You will also implement two learning curve graphs displayed at the end of the training phase (you are free to use any library you want for this purpose).

For `example`:

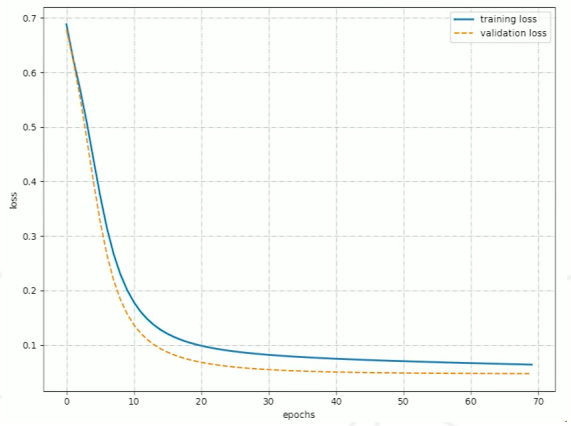


Figure IV.1: Loss

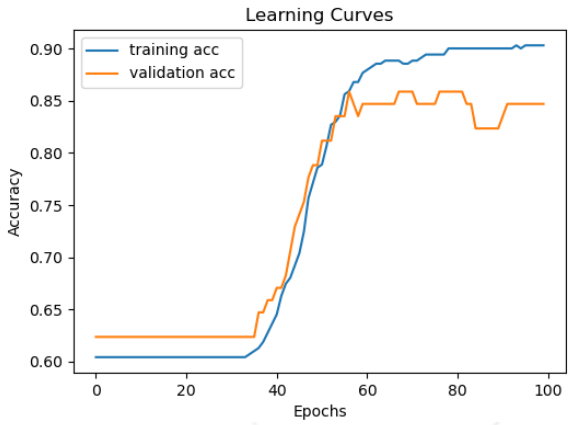


Figure IV.2: Accuracy

IV.4 Submission

You will submit three programs:

- A program to separate the data set into two parts, one for training and the other for validation
- A training program
- A prediction program

(or you can submit a single program with an option to switch between the three phases)

To visualize your model performances during training, you will display at each epoch the training and validation metrics.

For `example`:

```
python mlp.py --dataset data_training.csv
x_train shape : (342, 30)
x_valid shape : (85, 30)
epoch 01/70 - loss: 0.6882 - val_loss: 0.6788
...
epoch 39/70 - loss: 0.0750 - val_loss: 0.0406
epoch 40/70 - loss: 0.0749 - val_loss: 0.0404
epoch 41/70 - loss: 0.0747 - val_loss: 0.0400
...
epoch 70/70 - loss: 0.0640 - val_loss: 0.0474
> saving model './saved_model.npy' to disk...
```

- For the separate program you are allowed to use a seed to obtain a repeatable result, because a lot of random factors come into play (the weights and bias initialization for example)
- The training program will use `backpropagation` and `gradient descent` to learn on the training dataset and will save the model (network topology and weights) at the end of its execution.
- The prediction program will load the weights learned in the previous phase, perform a prediction on a given set (which will also be loaded), then evaluate it using the binary cross-entropy error function :

  $E = -\frac{1}{N}\sum_{n=1}^{N}[y_n \log p_n + (1-y_n)\log(1-p_n)]$

Chapter V

Bonus part

The bonus part will be evaluated only if the mandatory part was perfectly done. You are free to implement any functionalities that you think could be interesting. Nevertheless, here is an non-exhaustive list of bonuses :

- A more complex optimization function (for example : nesterov momentum, RMSprop, Adam, ...).
- A display of multiple learning curves on the same graph (really useful to compare different models).
- An historic of the metric obtained during training.
- The implementation of early stopping.
- Evaluate the learning phase with multiple metrics.

The bonus part will only be assessed if the mandatory part is PERFECT. Perfect means the mandatory part has been integrally done and works without malfunctioning. If you have not passed ALL the mandatory requirements, your bonus part will not be evaluated at all.

Chapter VI

Submission and peer-evaluation

Turn in your assignment in your `Git` repository as usual. Only the work inside your repository will be evaluated during the defense. Don't hesitate to double check the names of your folders and files to ensure they are correct.