Documentation pour la classe Neuron obtenable avec help(Neuron) :

class Neuron(builtins.object)

| fields:

| input\_size: int

| beta: list of the coefficitents used in the linear combination of the

| outputs of the previous layer

|

| Methods defined here:

|

| \_\_init\_\_(self, input\_size)

| Creates a Neuron object with random coefficients

| ----

| input:

| input\_size: int -> n\_c where c is the layer of the neuron

| ----

| output:

| void

|

| comb\_lin(self, Zc)

| Returns the sum of beta \* Z

| ----

| Input:

| Zc: array of length n\_c -> outputs of the previous layer

| ----

| output:

| out: float -> sum of beta \* Z

|

| compute\_output(self, Zc)

| Given the outputs of the previous layer, computes the output of

| this neuron

| ----

| input:

| Zc : array of length n\_c -> outputs of the previous layer

| ----

| output:

| out : float -> output of the neuron

Documentation pour la classe NeuralNetwork obtenable avec help(NeuralNetwork) :

class NeuralNetwork(builtins.object)

| fields:

| format: array of integers of size C -> (n\_c)\_c

| neuron\_layers: array of arrays ofNeurons array length C. The c th layer has length n\_c

| -> a line represents a layer of neurons

| Z\_layers: float array array -> stores the current output of each Neurons

| learning\_Rate: float -> learning Rate used for backpropagation

| current\_input: float array of size p -> the input being computed

| errors: float array of size n\_C -> the error of each line

| derivatives: float array -> derivatives[c][k][j] = dR / dbeta\_(j, k)^c

|

| Methods defined here:

|

| \_\_init\_\_(self, format, p)

| Creates a NeuralNetwork object with random coefficients and a given

| size for each layer

| ----

| input:

| format: array of integer of size C -> (n\_c)\_c

| p: int -> the numberof column used as input of the network

| ----

| output:

| void

|

| compute\_all(self, database, outputs)

| Make the every line go through the network, storing the errors of each one

| ----

| input:

| database: array of shape (N, p) -> the training database

| outputs: array of shape (N, n\_C) -> the expected outputs (y)

| ----

| output:

| void

|

| compute\_derivatives(self, expected\_output)

| Adds the derivative of R\_i with respect of every coefficient to the

| derivative matrix

| ----

| input:

| expected\_output: float array of length n\_C -> the outputs expected

| for the current inputs

| ----

| output:

| void

|

| compute\_error(self, expected\_output)

| Returns the current error

| ----

| input:

| expected\_output: float array of size n\_C -> y

| ----

| output:

| res: float -> R\_i(theta)

|

| compute\_one(self, input)

| Make the input go through the network and stores the outputs of each layer

| ----

| input:

| input: array of floats of size p -> (x\_i)\_i

| ----

| output:

| void

|

| deriv\_Z(self, m, cz, j, k, cb)

| Returns the derivative of Z\_m^cz with respect to beta\_(j, k)^cb

| ----

| input:

| m, cz, j, k, cb: int

| ----

| output:

| res: float -> the derivative

|

| deriv\_error\_i(self, j, k, c, expected\_output)

| Returns the derivative of R\_i with respect to beta\_(j, k)^c

| ----

| input:

| j, k, c: int

| expected\_output: float array of length n\_C -> the outputs expected

| for the current inputs

| ----

| output:

| res: float -> the derivative

|

| predict(self, database)

| Predicts the outputs for each line of the database

| ---

| input:

| database: float array of shape (N, p)

| ----

| output:

| prediction: float array of shape (N, n\_C)

|

| total\_error(self)

| Returns the error of one line

| ----

| input:

| void

| ----

| output:

| res: float -> R = \sum\_i R\_i

|

| train(self, database, outputs, n)

| Trains the network on the database

| ----

| input:

| database: float array of shape (N, p)

| outputs: float array of shape (N, n\_C)

| n: int -> how many times the database will go through the network

| ----

| output:

| error\_list: float array of length n -> the error after each turn

|

| update\_coeff(self)

| Update every coefficient of the network using backpropagation

| ----

| input:

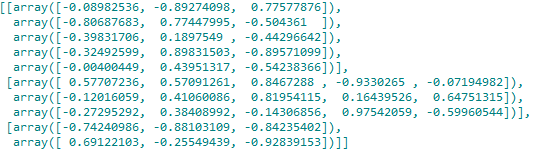
| void

| ----

| output:

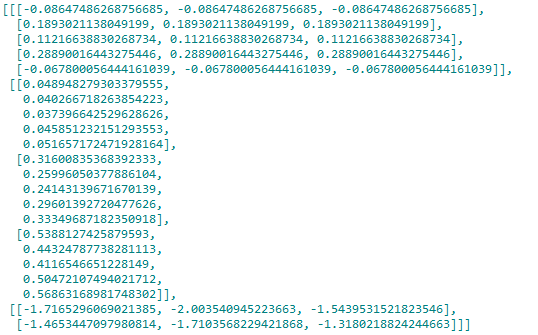
| void

Bon fonctionnement :

Coefficients initiaux, générés aléatoirement dans , pour un réseau avec des entrées de taille 3, une couche de 5 neurones, une de 3 neurones puis deux sorties.   
Les rectangles sont pour la suite.

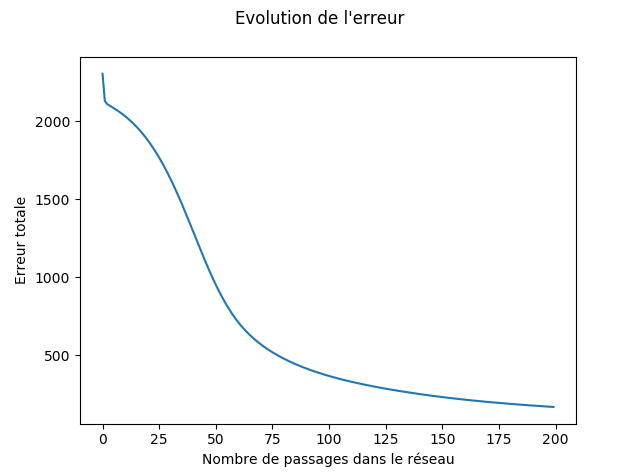


On fait passer l’entrée dans le réseau.   
Ligne 1 et 2 : sortie des neurones des couches 1 et 2.   
Ligne 3 : sortie du réseau de neurone (on applique pas sigma).   
Rectangles pour ce qui suit



Les dérivées par rapport à chaque coefficient (première couche en haut, dernière en bas).   
Par exemple, on a que par application de la formule. On a (termes encadrés en rouge avant), on trouve bien le même résultat.

De même, on a



Évolution de l’erreur sur un échantillon de 4500 lignes de la base de données de spam (après mise à l’échelle), pour 200 passages dans un réseau , avec un learning rate constant de .   
Le procédé a pris plusieurs heures, et la prédiction sur les 101 lignes restant a montré un taux de réussite de 97%.