```
# Dans cette partie, on utilise le module csv pour lire les données d'une base de données de test.
# Cette base est utilisée pour entraîner/tester un réseau de neurone à apprentissage supervisé.
fichier = open("iris.csv")
import csv
lecteur = csv.reader(fichier)
for ligne in lecteur:
  if(lecteur.line_num == 1):
    n = int(ligne[0])
    p = int(ligne[1])
    c = int(ligne[2])
    x = []
    d = []
  else:
    x.append([float(ligne[j]) for j in range(p)])
    tmp = [0 for i in range(c)]
    tmp[int(ligne[p])-1] = 1
    d.append(tmp)

print(x[50])
print(d[50])

# Implémentation d'un neurone formel
from random import random
from math import exp
class Neurone:
  def __init__(self, nbrPoids):
    self.nbrPoids = nbrPoids
    self.sortie = random()
    self.poidsSynap = [random() for i in range(nbrPoids)]
    self.biais = random()
    self.entree = []
    self.delta = 0.
  def chargeEnt(self, vectEnt):
    self.entree = [vectEnt[i] for i in range(self.nbrPoids)]
  def calculSortie(self): #Nécissite l'exécution de la fonction chargeEnt
    self.sortie = self.biais
    for i in range(self.nbrPoids):
      self.sortie += self.poidsSynap[i]*self.entree[i]
    self.sortie = 1./(1.+exp(-self.sortie))
  def ajustPoids(self, eta, delta):
    self.delta = delta
    for i in range(self.nbrPoids):
      self.poidsSynap[i] += eta*self.delta*self.entree[i]
    self.biais += eta*delta


nrn = Neurone(4)
print(x[20])
nrn.chargeEnt(x[20])
nrn.calculSortie()

# Implémentation d'une couche neuronale
class Couche:
  def __init__(self, nbrNrn, nbrPoids):
    self.nbrNrn = nbrNrn
    self.nbrPoids = nbrPoids
    self.neurone = [Neurone(nbrPoids) for i in range(nbrNrn)]
  def chargeEnt(self, vectEnt):
    for i in range(self.nbrNrn):
      self.neurone[i].chargeEnt(vectEnt)
  def calculSortie(self): # Prérequis: exécution de la fonction chargerEnt
    for i in range(self.nbrNrn):
      self.neurone[i].calculSortie()
  def affichSortie(self):
    print([self.neurone[i].sortie for i in range(self.nbrNrn)])
  def ajustPoids(self, eta, delta):
    for i in range(self.nbrNrn):
      self.neurone[i].ajustPoids(eta, delta[i])
cch = Couche(3,p)
for i in range(3):
  print(str(cch.neurone[i].poidsSynap) + " " + str(cch.neurone[i].biais))
print()
print(x[60])
print()
cch.chargeEnt(x[0])
cch.calculSortie()
cch.affichSortie()
for i in range(1000):
  cch.calculSortie()
  for j in range(cch.nbrNrn):
    delta = (d[0][j] - cch.neurone[j].sortie)*cch.neurone[j].sortie*(1. - cch.neurone[j].sortie)
cch.neurone[j].ajustPoids(0.2, delta)
print(nrn.sortie)
class PMC:
  def __init__(self, nbrCch, nbrNrn, nbrPoids): #nbrNrn: tab, nbrPoids: tab
    #la première composante du vect nbrNrn contient le nombre de neurones de la première couche
    self.couche = [Couche(nbrNrn[i], nbrPoids[i]) for i in range(nbrCch)]
    self.nbrCch = nbrCch
    self.nbrNrn = [nbrNrn[i] for i in range(self.nbrCch)]
    self.nbrPoids = [nbrPoids[i] for i in range(self.nbrCch)]
  def propager(self, vectEnt): # c'est calculerSortie pour le réseau de neurones
    self.couche[0].chargeEnt(vectEnt)
    self.couche[0].calculSortie()
    for i in range(1, self.nbrCch):
      sortCchPrec = [self.couche[i-1].neurone[j].sortie for j in range(self.nbrNrn[i-1])]
      self.couche[i].chargeEnt(sortCchPrec)
      self.couche[i].calculSortie()
  def affichSortie(self):
    self.couche[self.nbrCch-1].affichSortie()
  def entrainer(self, X, D, eta, tmax):
    for r in range(tmax):
      for q in range(len(X)):
        # Propager les signaux d'entrée pour chaque exemple
        self.propager(X[q])
        # Ajuster les poids synaptiques des neurones de la couche de sortie
        delta = [(D[q][j] - self.couche[self.nbrCch-1].neurone[j].sortie)*self.couche[self.nbrCch-1].
neurone[j].sortie*(1. - self.couche[self.nbrCch-1].neurone[j].sortie) for j in range(self.couche[self.nbrCch-1].nbrNrn)]
        for j in range(self.couche[self.nbrCch-1].nbrNrn):
          self.couche[self.nbrCch-1].neurone[j].ajustPoids(eta, delta[j])
        # Ajuster les poids synaptiques des neurones des couches cachées et de la couche d'entrée
        for s in range(self.nbrCch-2, -1, -1):
          delta = [random() for i  in range(self.couche[s].nbrNrn)]
          for k in range(self.couche[s].nbrNrn):
            delta[k] = 0.
            for i in range(self.couche[s+1].nbrNrn):
              delta[k] += self.couche[s+1].neurone[i].poidsSynap[k]*self.couche[s+1].neurone[i].delta
            delta[k] *= self.couche[s].neurone[k].sortie*(1. - self.couche[s].neurone[k].sortie)
          for j in range(self.couche[s].nbrNrn):
            self.couche[s].neurone[j].ajustPoids(eta, delta[j])
pmc = PMC(2, [3, c], [p, 3])
pmc.entrainer(x, d, 0.2, 10000)
```