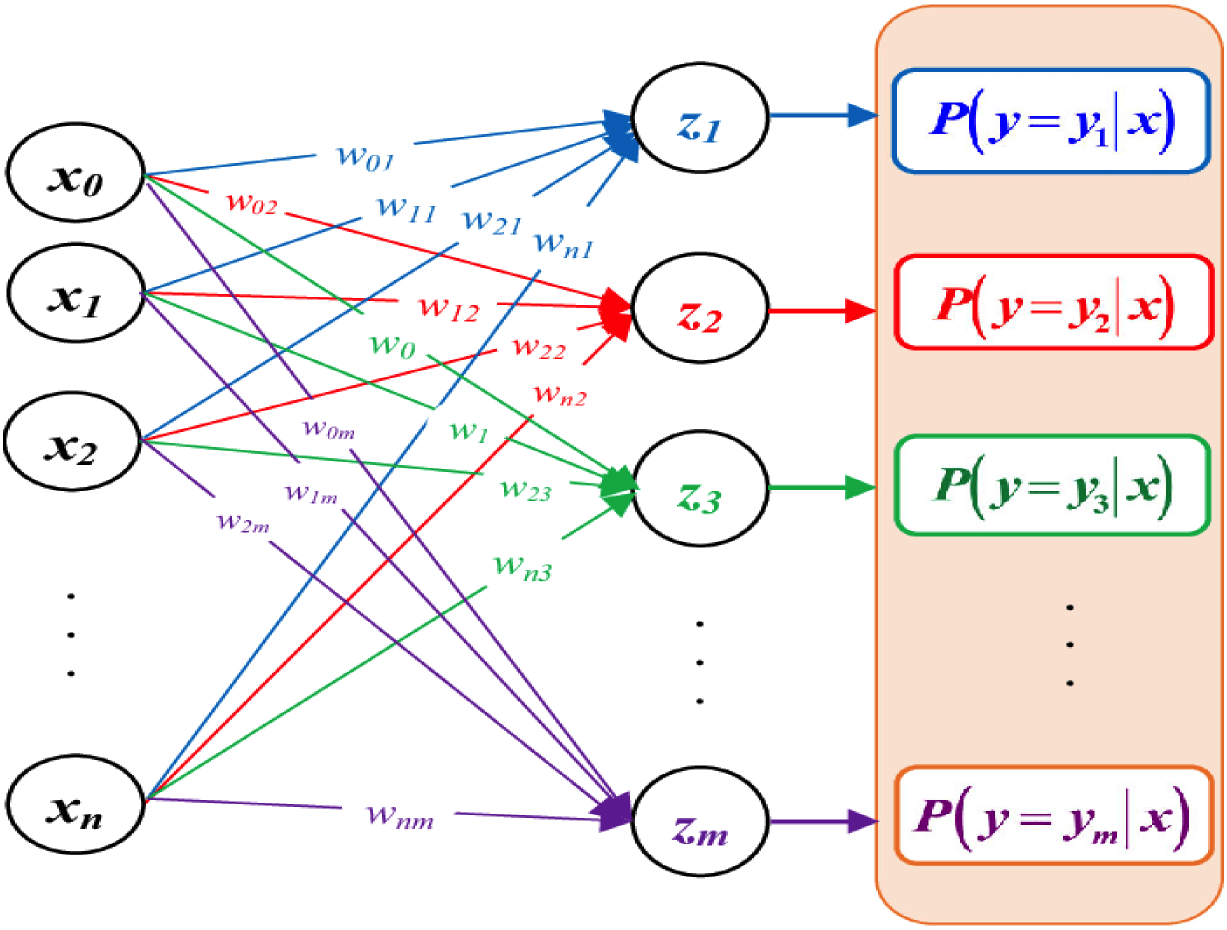
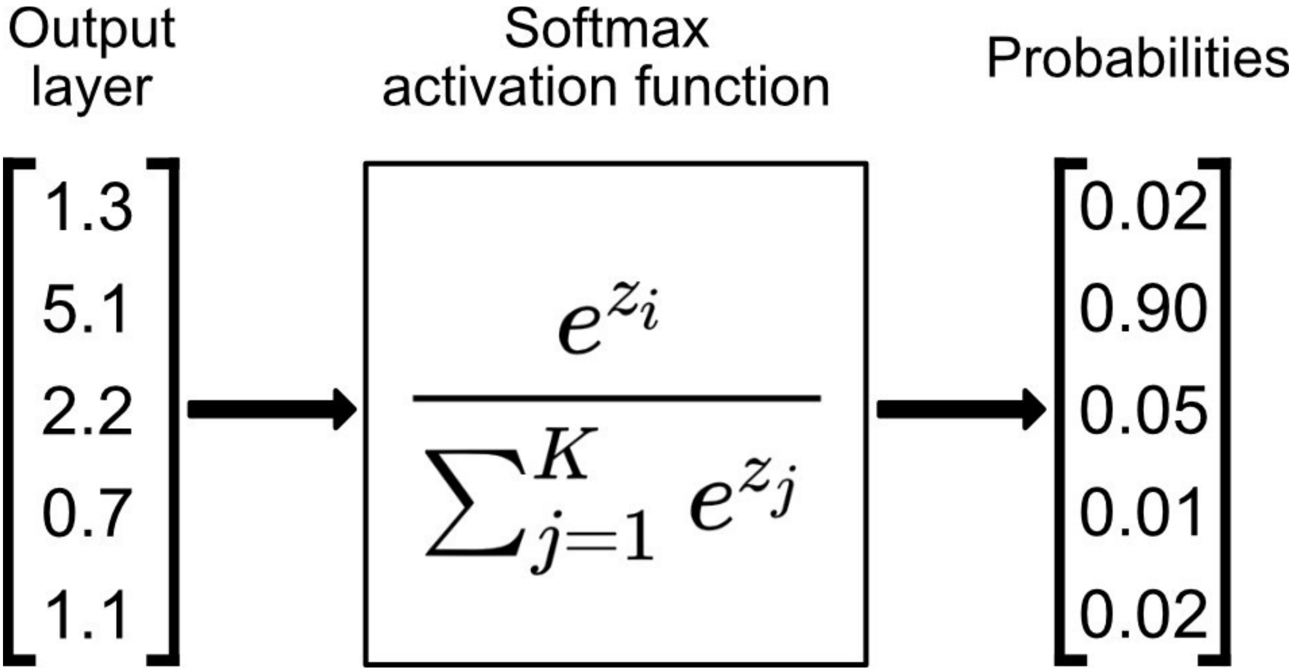
La fonction SoftMax et la classification Multi_Classe




Output layer
Softmax activation function
Probabilities
1.3
5.1
2.2
0.7
1.1
$\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$
0.02
0.90
0.05
0.01
0.02

Trame (python) de l'algorithme de retro_propagation du gradient

```
import numpy as np
import time

from keras.datasets import mnist
from keras.utils import np_utils

K=10;
(X_train, Y_train), (X_test, Y_test) = mnist.load_data()
X_train=X_train.reshape(60000, 784)
X_test=X_test.reshape(10000, 784)
X_train=X_train.astype('float32')
X_test=X_test.astype('float32')
X_train/=255
X_test/=255
print(X_train.shape[0], 'train samples')
print(X_test.shape[0], 'test samples')

#convertir class vecteur to Multi_Class matrices
Y_train = np_utils.to_categorical(Y_train, K)
Y_test = np_utils.to_categorical(Y_test, K)

def softmax(X):
   # input matrix X of size Nbxd
   E=np.exp(X)
   return (E.T/np.sum(E,axis=1)).T

def forward (images, W, b):
   pred=np.matmul(images, W) + b
   return softmax(pred)

def accuracy (W,b,images,labels):
   pred=forward(images, W, b)
   return np.where(pred.argmax(axis=1) != labels.argmax(axis=1), 0., 1.).mean()*100.0

d=X_train.shape[1]
N=X_train.shape[0]
numIt=20

W=np.zeros((d,K))
biais=np.zeros((1,K))
eta=1e-1

batch_size=100
nb_batches=int(float(N)/batch_size)

gradW=np.zeros((d,K))
gradb=np.zeros((1,K))

for it in range(numIt):
   for b in range(nb_batches):
    # Forward
    pred=forward(X_train[b*batch_size:(b+1)*batch_size,:], W, biais)
    val=pred-Y_train[b*batch_size:(b+1)*batch_size,:]
    #Backward
    gradW=1.0/batch_size*np.matmul(np.transpose(X_train[b*batch_size:(b+1)*batch_size,:]),
       val)
    gradb=1.0/batch_size*(val.sum(axis=0)).reshape((1,10))

    W=W-eta*gradW
    biais=biais-eta*gradb

    print ("epoch ", it, "accuracy train=", accuracy(W, biais, X_train, Y_train), "accuracy test=",
accuracy(W,biais,X_test, Y_test)
      )
```