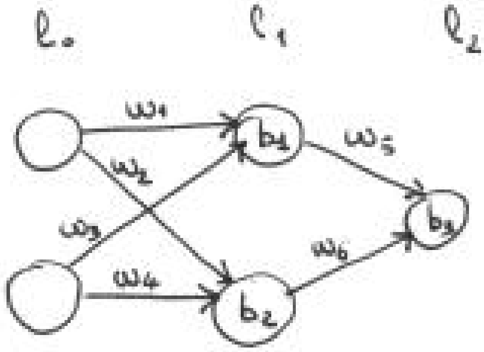
Backpropagation (algo)

Soit un réseau de topologie par couche suivant : R



on définit 2 listes :

$$\text{biases} = \left(\begin{pmatrix} b_1 \\ b_2 \end{pmatrix}, \begin{pmatrix} b_3 \end{pmatrix} \right)$$

$$\text{weights} = \left(\begin{pmatrix} w_1 & w_2 \\ w_3 & w_4 \end{pmatrix}, \begin{pmatrix} w_5 & w_6 \end{pmatrix} \right)$$

Ainsi que la liste $\Omega = \left(\left(\begin{pmatrix} b_1 \\ b_2 \end{pmatrix}, \begin{pmatrix} w_1 & w_2 \\ w_3 & w_4 \end{pmatrix} \right), \left(\begin{pmatrix} b_3 \end{pmatrix}, \begin{pmatrix} w_5 & w_5 \end{pmatrix} \right) \right)$

Celle-ci représente l'ensemble du réseau R.

Pour débuter on va initialiser les b_i et w_i avec des valeur aléatoires, il existe des manière plus efficace pour procéder à cette initialisation.

1) feed forward

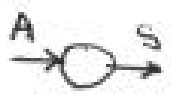
La première étape consiste à "nourrir" le réseau avec un vecteur d'entré x possédant le même nombre de composantes qu'il y a de neurones d'entrée sur le réseau.

Ici, il n'y a que 2 neurones donc $x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$

→ déroulement de l'algo

$a = x$; la couche d'entré est formé de neurones neutre $S = A$

donc la sortie l_0 est x



activations $[x]$; on ajoute x à liste des vecteurs d'activation

$zs = [\,]$; on créer une liste permettant de recueillir les différentes sorties des couches de neurones

on définit deux fct° :

$$\sigma(z) = \frac{1}{1+e^{-z}} \quad \text{et} \quad \sigma'(z) = \frac{\sigma(z)}{1-\sigma(z)}$$

Pour tous les b, w de Ω

$z = w \cdot a + b$; $\Rightarrow \begin{cases} \begin{pmatrix} w_1 & w_2 \\ w_3 & w_4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} = \begin{pmatrix} w_1 x_1 + w_2 x_2 + b_1 \\ w_3 x_1 + w_4 x_2 + b_2 \end{pmatrix} = z_1 = \begin{pmatrix} z_{1,1} \\ z_{1,2} \end{pmatrix} \text{ (couche 1)} \\ \begin{pmatrix} w_5 & w_6 \end{pmatrix} \begin{pmatrix} \sigma(z_{1,1}) \\ \sigma(z_{1,2}) \end{pmatrix} + b_3 = w_5 \sigma(z_{1,1}) + w_6 \sigma(z_{1,2}) + b_3 = z_2 \text{ (couche 2)} \end{cases}$

zs.append(z) ;

$a = \sigma(z)$;

activations append(a)

Donc au final nous obtenons 2 listes.

$zs = [z_1 ; z_2]$; z_1 et z_2 sont des matrices

et

activations $= [x ; \sigma(z_1) ; \sigma(z_2)]$; $\sigma(z)$ est appliqué sur chaque composantes de z_i

2) Vérifications

Maintenant que le vecteurs d'entré x s'est propagé jusqu'à la couche de sortie sous la forme du vecteur $\sigma(z_2)$

Il faut vérifier si la ~~valeur~~ résultante est correct ou non.

Pour cela on utilise un second vecteur y ~~$= (y_1)$~~ possédant le même nombre de composantes qu'il y a de ~~vecteurs~~ neurones de sortie. Ici $y = (y_1)$

On appelle la différence δ, elle mesure l'écart entre ~~ce~~ qui est attendu et ce que l'on désire.

$\delta = (\text{activations}[-1] - y) \; \sigma'(zs[-1])$; $\Rightarrow \delta = (\sigma(z_2) - y) \, \sigma'(z_2)$

Il est prouvé mathématiquement que ~~cette~~ ~~valeur~~ vecteur δ permet d'améliorer les résultats des neurones de sortie.

3) Initialisation de la backpropagation

On définit 2 listes :

$$\nabla_{b_0} = \left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, (0) \right) \quad \text{et} \quad \nabla_{w_0} = \left(\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}, (0\ 0) \right)$$

La backpropagation comme son nom l'indique a pour but de propager une modification des sorties vers les entrées. Contrairement au feedforward qui propage les entrées à travers les couches successives pour atteindre les neurones de sortie.

Le vecteur δ calculé précédemment sera le pointe départ pour modifier l'ensemble du réseau.

Soit :

$\nabla_b[-1] = \delta;$

$$\nabla w[-1] = \delta \cdot (\text{activations}[-2])^T ; \Rightarrow \delta\, \sigma(z_1)^T = \delta \cdot \begin{pmatrix} \sigma(z_{1,1}) \\ \sigma(z_{1,2}) \end{pmatrix}^T$$
$$\Rightarrow \delta\, (\sigma(z_{1,1})\ \ \sigma(z_{1,2}))$$
$$\Rightarrow (\delta\, \sigma(z_{1,1})\ \ \delta\, \sigma(z_{1,2})) = \nabla w[-1]$$

4) Back propagation

Pour l'ensemble des layer qui ne sont ni des entrées ni des sorties ; $l = [2]$

$z = z_s[-l]; \Rightarrow z = z_s[-2] = z = z_1$ (avant dernier élément de z_s)

$w = \text{weights}[-l+1] ; \Rightarrow w = \text{weights}[-2+1] = \text{weights}[-1] = (w_5\ \ w_6)$

$$\delta = w^T \cdot \delta \cdot \sigma'(z) ; \Rightarrow \left\{ \begin{aligned} \delta &= \left((w_5\, w_6)^T \cdot \delta \right) \begin{pmatrix} \sigma'(z_{1,1}) \\ \sigma'(z_{1,2}) \end{pmatrix} \\ &= \begin{pmatrix} w_5 \\ w_6 \end{pmatrix} \cdot \delta \begin{pmatrix} \sigma'(z_{1,1}) \\ \sigma'(z_{1,2}) \end{pmatrix} \\ &= \begin{pmatrix} \delta w_5 & \delta w_6 \end{pmatrix} \begin{pmatrix} \sigma'(z_{1,1}) \\ \sigma'(z_{1,2}) \end{pmatrix} = \begin{pmatrix} \delta w_5\, \sigma'(z_{1,1}) \\ \delta w_6\, \sigma'(z_{1,2}) \end{pmatrix} \end{aligned} \right.$$

$\nabla b[-l] = \delta$

$$= \nabla_b[-2]$$
$$= \begin{pmatrix} \delta_1 \\ \delta_2 \end{pmatrix}$$

$$\nabla w[-l] = \delta \left(\text{activations}[-l-1]\right)^T ; \Rightarrow \begin{cases} \text{activations}[-2-1] = \text{activations}[-3] \\ \qquad = x \\ \nabla w[2] = \delta x^T = \delta \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}^T \\ = \begin{pmatrix} \delta_1 \\ \delta_2 \end{pmatrix} \begin{pmatrix} x_1 & x_2 \end{pmatrix} \\ = \begin{pmatrix} \delta_1 x_1 & \delta_1 x_2 \\ \delta_2 x_1 & \delta_2 x_2 \end{pmatrix} \end{cases}$$

Une fois la backpropagation effectuée, nous obtenons 2 liste

$$\nabla b = \left(\begin{pmatrix} \delta_1 \\ \delta_2 \end{pmatrix}, (\delta) \right) \quad \text{et} \quad \nabla w = \left(\begin{pmatrix} \delta_1 x_1 & \delta_1 x_2 \\ \delta_2 x_1 & \delta_2 x_2 \end{pmatrix}, \begin{pmatrix} \delta\sigma(S_{11}) & \delta\sigma(S_{12}) \end{pmatrix} \right)$$

Celles ci seront utiliser à chaque envoie de résultat pour modifier le réseau R. Et le rendre de plus en plus adapté à nos besoins.

On appelle ces 2 liste les liste de différences de poids et de biais par la suite on les renomme donc ainsi :

$$\nabla b \rightarrow \delta \nabla b \quad \text{et} \quad \nabla w \longrightarrow \delta \nabla w$$

5) Apprentissage par l'erreur

Au départ : $\nabla w = \nabla w_0$ et $\nabla b = \nabla b_0$

Pour tous les layer l excepté celui d'entrée :

$$\nabla b[l] = \nabla b[l] + \delta \nabla b[l]$$
$$\nabla w[l] = \nabla w[l] + \delta \nabla w[l]$$

~~Pour tous les poids w de weights~~.

On définit deux variables

m : nombre de valeur d'entraînement
c : coefficient d'apprentissage (règle la réactivité du réseau)

Pour tous les poids w de weights :

$$w = w - \frac{c}{m} \nabla w[\cancel{l}]_j \Rightarrow \begin{cases} W_1 = \begin{pmatrix} w_1 & w_2 \\ w_3 & w_4 \end{pmatrix} - \frac{c}{m} \begin{pmatrix} \delta_1 x_1 & \delta_1 x_2 \\ \delta_2 x_1 & \delta_2 x_2 \end{pmatrix} \\ W_2 = \begin{pmatrix} w_5 & w_6 \end{pmatrix} - \frac{c}{m} \begin{pmatrix} \delta\sigma(z_{1,1}) & \delta\sigma(z_{1,2}) \end{pmatrix} \end{cases}$$

Pour tous les biais b de biases

$$b = b - \frac{c}{m} \nabla b[\cancel{l}]_j \Rightarrow \begin{cases} B_1 = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} - \frac{c}{m} \begin{pmatrix} \delta_1 \\ \delta_2 \end{pmatrix} \\ B_2 = b_3 - \frac{c}{m} \delta \end{cases}$$

Et ceux pour l'ensemble du jeu de test.