Réseaux de neurones simples et Deep learning

Soit une image de (28×28) pixels. On represente cette image comme un élément de $[0,1]^{28\times 28}$, soit un ~~784~~ uplet de valeurs dans $[0,1]$.

l'image represente au plus un élément de l'ensemble des caractères hexadécimaux.

$$\{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F\} = C$$

on dit que ces 16 éléments sont les classes du problème.
On suppose qu'il existe une application $f : [0,1]^{28\times 28} \longrightarrow C$
un reseau neuronal est en faite une approximation de f.

- Fonctionnement d'un réseau de neurone standard : Forward pass

Un réseau de neurone est une composition de fonctions linéaires et non linéaires. Un neurone standard recoit un set d'échantillons S chacun contenant F caractéristiques.

$$\begin{array}{c}
\quad F \\
S\begin{bmatrix} x_{11} & \cdots & x_{1F} \\ \vdots & \ddots & \vdots \\ \vdots & & \vdots \\ x_{m1} & \cdots & x_{mF} \end{bmatrix} \\
\underbrace{\quad}_{m1} \qquad \underbrace{\quad}_{mF}
\end{array}$$

- Un ensemble de neurones constituent une couche. Un neurone recevant une matrice $(S \times F)$ nous renseigne sur les couches avant : F neurones.

Pour chaque neurone, on associe une matrice de poids. W, et un vecteur de bias B. Il est nécéssaire que W soit de dimensions $(F\times 1)$ et B soit de dimension $(S \times 1)$.

le neurone effectue les opérations matriciels suivantes :

$XW + B = Z$. une fois Z calculé, on applique une fonction non linéaire dite "Activation" à Z, élément par élément. Z est de dimension $(S\times 1)$ Par exemple : tanh, sigmoid $\left(\frac{1}{1+e^{-x}}\right)$, ReLU ... etc. Il est nécéssaire (ou plutot fortement conseillé) que la fonction d'activation soit monotone et injective (on authorise toutefois 0 en valeur multiple).
S'il y'a n neurones dans un niveau, les neurones de la couche suivantes recoivent des matrices de dimension $(S \times n)$

$$S\left\{\underbrace{\begin{bmatrix} x_{11}, & x_{12}, & \dots, & x_{1F} \\ \vdots & & & \vdots \\ \vdots & & \ddots & \vdots \\ x_{S1}, & x_{S2}, & \dots, & x_{SF} \end{bmatrix}}_{F}\right. \times F\left\{\underbrace{\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_F \end{bmatrix}}_{1}\right. + S\underbrace{\begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_S \end{bmatrix}}_{1} = S\underbrace{\begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_S \end{bmatrix}}_{1} \Rightarrow \text{Activation}\begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_S \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} \text{Activation}(z_1) \\ \vdots \\ \text{Activation}(z_S) \end{bmatrix} \quad (S \times 1)$$

soit um meurone de la couche suivante. et soit n neurones dans la couche qui le précède. le neurone reçoit une matrice de dimension $(s \times n)$: c'est la concaténation des s vecteurs colonnes renvoyés par chaque neurone de la couche anterieure.
Le forward pass se termine par une couche contenant 16 neurones; soit la cardinalité de nos classes. Cette couche renvoie une matrice de dimension $(S \times 16)$

$$S\left[\begin{array}{c}c_{00}, c_{01}, c_{02}, \dots, c_{0\,15} \\ \vdots \quad \ddots \quad \vdots \\ c_{s0}, c_{s1}, c_{s2}, \dots, c_{s\,15}\end{array}\right]_{16}$$

chaque C_{ij} est une probabilité que l'échantillon $i \in S$ soit de classe $j \in C$.
On suppose que pour chaque echantillon, la vraie classe soit connue, on peut alors regrouper ces classes en une matrice Y de même dimension

$$P = \begin{bmatrix} 0, 1, 0, \cdots, 0 \\ 0, 0, 1, \cdots, 0 \\ 1, 0, \cdots, 0, 0 \\ \vdots \\ 0, 1, \cdots, 0, 0 \end{bmatrix}$$
$(S \times 16)$

si P est la matrice des prédictions, et Y la matrice d'observation, on calcule le taux d'erreur, une valeur numérique donnée par une fonction cout opérant ligne à ligne. on obtient un vecteur d'erreurs : chaque erreur correspond à un échantillon

Fonction cout 1 : MSE : sur L_i : $\sum_{j=0}^{15} (Y_{ij} - P_{ij})^2$ (borne sup inclusе)

Fonction cout 2 : CE : sur L_i : $\sum_{j=0}^{15} -Y_{ij} \times \ln(P_{ij}) - (1 - Y_{ij}) \times \ln(1 - P_{ij})$

la fonction CE (cross entropy loss) est plus efficace sur des lignes ayant été altérées par la fonction softmax : $\text{Softmax}(L_i) = \frac{e^{L_{ij}}}{\sum e^{L_{im}}}$

une fois le vecteur d'erreur calculé, de dimension $(S \times 1)$, le forward pass est terminé. $(CES = CE(\text{softmax}))$

• backward pass
Rappel mathématique : • le gradient d'une fonction $g(x_1, x_2, \dots, x_m)$ en x est un vecteur des dérivées partiels de g en x : $\frac{\partial g(x)}{\partial x} = \left[\frac{\partial g(x)}{\partial x_1}, \frac{\partial g(x)}{\partial x_2}, \dots, \frac{\partial g(x)}{\partial x_m}\right]$

• la règle de la chaine : $\frac{d(g(h(w)))}{dw} = \frac{d(g(h(w)))}{d\,h(w)} \times \frac{d\,h(w)}{dw}$

On avait dit qu'un réseau neuronal est une approximation de f, la fonction de $[0,1]^{28 \times 28}$ dans C. le taux d'erreur calculée par CES est une mesure de cette approximation. De plus cette approximation n'est autre qu'une composition de fonctions, avec chaque neurone étant : Activation$(XW+B)$ ou X est un set de vecteurs lignes contenant les resultats des neurones de la couche anterieure. On discute à présent l'algorithme de backpropagation, qui permet d'ajuster tous les W et B de chaque neurone du réseau neuronale.
Notons déja que la fonction composée $CES = CE(\text{softmax})$ est une application de $[0,1]^{16} \times [0,1]^{16}$ dans $\mathbb{R}$ opérant sur les lignes de Y et P resp.
on nomme $X = (x_1, x_2, \dots, x_{16}) \in [0,1]^{16}$ les élements de la ligne P_i
$Z = Y_i$

On calcule les dérivées partielles de CES par rapport à chaque $x_i \in X$

on trouve : $\dfrac{\partial CE(Z, softmax(X))}{\partial x_1} = softmax(x_1) - Z_1 = \dfrac{e^{x_1}}{e^{x_1} + \dots + e^{x_{16}}} - Z_1$

donc, pour la matrice P, le gradient de la ligne i est simplement $softmax(P_i) - Y_i$, et de manière plus générale, pour de multiples échantillons

$$\begin{bmatrix} \left[\frac{e^{c_{11}}}{\Sigma e^c} - P_{11}, \frac{e^{c_{12}}}{\Sigma e^c} - P_{12}, \dots, \frac{e^{c_{116}}}{\Sigma e^c} - P_{116} \right] \\ \vdots \qquad \vdots \qquad \ddots \qquad \vdots \\ \left[\frac{e^{c_{s1}}}{\Sigma e^c} - P_{s1}, \frac{e^{c_{s2}}}{\Sigma e^c} - P_{s2}, \dots, \frac{e^{c_{s16}}}{\Sigma e^c} - P_{s16} \right] \end{bmatrix}$$

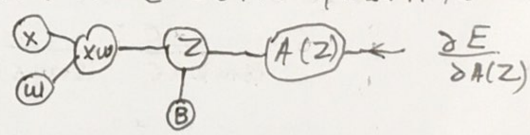
on nomme cette matrice la matrice des gradients des échantillons

Pour chaque neurone de la dernière couche, on ajuste les poids W et B

Par abstraction : le neurone N reçoit le vecteur colonne :

$$\begin{bmatrix} \frac{e^{c_1}}{\Sigma e^c} - P_{11} \\ \vdots \\ \frac{e^{c_{s1}}}{\Sigma e^c} - P_{s1} \end{bmatrix}$$ qu'on renomme Gl (gradient local) $\begin{bmatrix} Gl_1 \\ \vdots \\ Gl_s \end{bmatrix} = Gl$

on effectue la somme des Gl_i, ce qui donne un gradient global du neurone en question. on nomme cette quantité $\dfrac{\partial E}{\partial A(Z)}$; graphe de computation :



l'algorithme de back propagation se déroule ainsi

$\dfrac{\partial E}{\partial A(Z)}$ donnée

$\dfrac{\partial A(Z)}{\partial Z} = A'(Z)$

$\dfrac{\partial Z}{\partial B} = 1$

$\dfrac{\partial Z}{\partial XW} = 1$

$\dfrac{\partial XW}{\partial X} = W$

$\dfrac{\partial XW}{\partial W} = X$

$$\frac{\partial E}{\partial W} = \frac{\partial E}{\partial A(Z)} \times \frac{\partial A(Z)}{\partial Z} \times \frac{\partial Z}{\partial XW} \times \frac{\partial XW}{\partial W} =: gw$$

ajustement $W = W - \alpha \times gw$ $\qquad \alpha$ = learning rate

$$\frac{\partial E}{\partial B} = \frac{\partial E}{\partial A(Z)} \times \frac{\partial A(Z)}{\partial Z} \times \frac{\partial Z}{\partial B} =: gb$$

$B = B - \alpha \times gb$ (ajustement)

à renvoyer à la couche anterieure

$$\frac{\partial E}{\partial X} = \frac{\partial E}{\partial A(Z)} \times \frac{\partial A(Z)}{\partial Z} \times \frac{\partial Z}{\partial XW} \times \frac{\partial XW}{\partial X}$$

on peut, étant donné une couche entière, exprimer ces opérations en termes d'opérations bien plus élémentaires, Transposition des vecteurs W et X.