- apprentissage supervisé :

$$\mathcal{D} = \left\{ \left(x^{(1)}, y^{(1)}\right), \left(x^{(2)}, y^{(2)}\right), \left(x^{(3)}, y^{(3)}\right), \dots \right\}$$

⇒ recherche d'un prédicteur minimisant la différence entre labels prédits et réels

⇒ necessite l'annotation de larges bases de données

- apprentissage non supervisé

inférence de l'info à partir d'observation uniquement

a/ réduction de données b/ extraction de caractéristiques

c/ clustering d/ estimation de densité

- Renforcement learning

apprentissage faiblement supervisé

- un trop petit nombre d'annotations (few shots)
  one shot learning
- Pas annotations sont trop bruitées ou imprécises

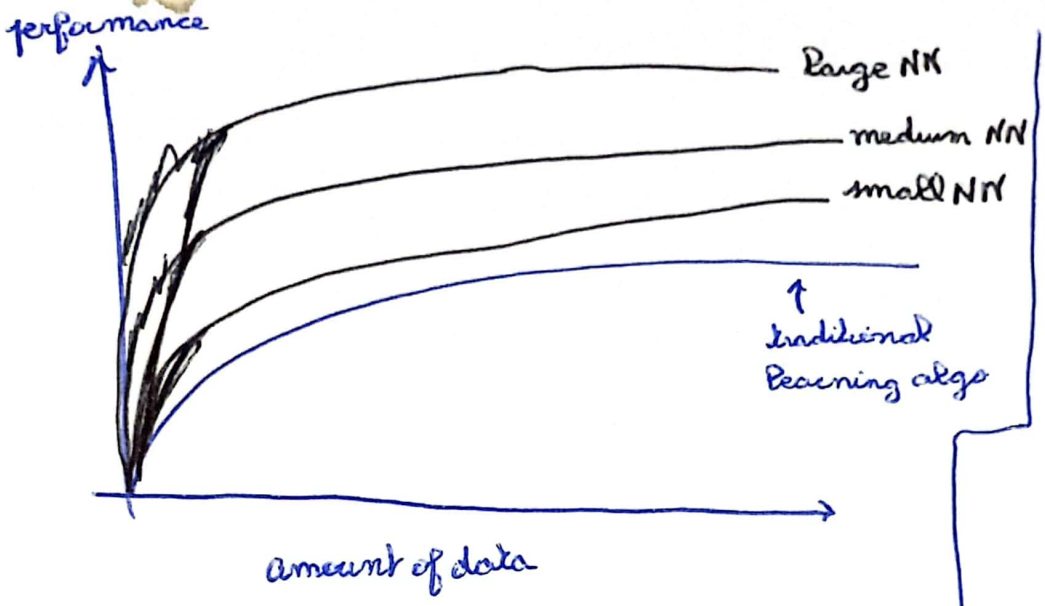
apprentissage semi supervisé

faible nombre de données annotées
+
grand nombre de données non annotées

apprentissage actif

ensemble de données dont seulement certains sont annotées
⇒ le modèle doit déterminer au travers d'un certain critère à définir quelles ~~sont~~ les données sont potentiellement lui fournir le plus d'infos pour évoluer vers de meilleures perfos
⇒ un expert peut annoter ces données

Finance
* prédire le cours d'une action
*



⇒ scale drives deep learning progress:
- data
- computation
- algorithms

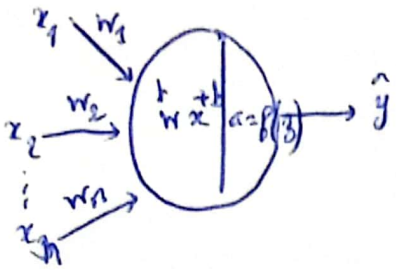
small training data:

to get better performance:

skills at handling features

Perceptrons

neuronne : triplet + poids synaptique / biais b
+ fonction d'activation



algorithme de perceptron monocouche

1/ initialisation des poids $w_j^{\{0\}}$

2/ presentation du vecteur d'entrée $x^{(1)}, \dots, x^{(m)}$
et du vecteur de sortie $y^{(1)}, \dots, y^{(m)}$

3/ calcul de la sortie predite

$$\hat{y}^{(i)\{k\}} = f\left(\sum_{j=1}^{n} w_j^{\{k\}} x_j^{(i)}\right)$$

$$J(w) = \sum_{i=1}^{m} \text{perte}\left(\hat{y}^{(i)\{k\}}, y^{(i)}\right)$$

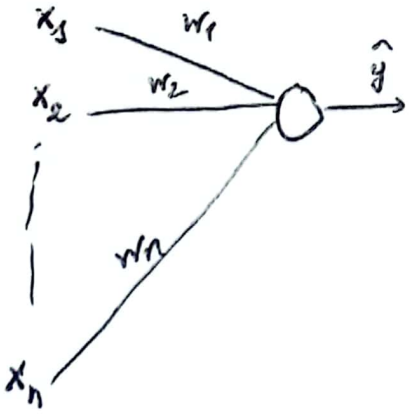
4/ mise à jour des poids : taux d'apprentissage

$$0 < \alpha \leq 1$$

$$w_j^{\{k+1\}} = w_j^{\{k\}} - \alpha \cdot \frac{\partial J}{\partial w_j}$$

5/ retourner en (2) jusqu'à la convergence

couche d'entrée



linear - activation + MSE ⇒ regression linéaire

sigmoid activation + cross entropy loss ⇒ regression logistique

Regression Linéaire

pour le $i^{ème}$ exemple :

$$\hat{y}^{(i)} = \theta_0 \cdot x^{(i)}$$ ~~→ equivalent~~

$$\mathcal{L}\left(y^{(i)}, \hat{y}^{(i)}\right) = \left(\hat{y}^{(i)} - y^{(i)}\right)^2$$

$$J(\theta) = \frac{1}{2} \sum_{i=1}^{m} \left(\theta_0 x^{(i)} - y^{(i)}\right)^2 : \boxed{\text{mean squared error}}$$

$X \in \mathbb{R}^{m \times (d+1)}$

$y \in \mathbb{R}^{m}$

m : nb d'exemples

d : nb of features

$\theta \in \mathbb{R}^{d+1}$

Fonction d'activation Linéaire

+

MSE

⇓

regression linéaire

On utilise un algo de descente de gradient pour éviter l'inversion de matrice qui est une opération très couteuse

One layer neuron networks (perceptrons)

1- binary classification

examples:
cat vs no cat
y: (1) (0)

dans les modèles Sklearn: on utilise la transposée de X
X. shape (m, n)

(X, y) $X \in \mathbb{R}^n$ $y \in \{0, 1\}$

m training examples:
$((x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)}))$

$m = m_{train}$

m_{test} : number of training examples

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix} \quad n$$

m columns

$$Y = \begin{bmatrix} y^{(1)}, y^{(2)}, \dots, y^{(m)} \end{bmatrix} \in \mathbb{R}^{1 \times m}$$

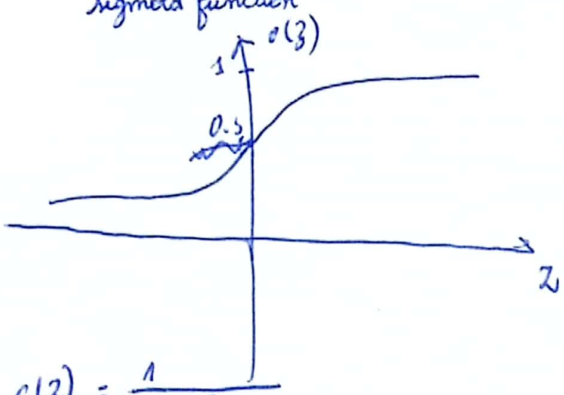
given x we want $\hat{y} = \mathbb{P}(y=1 \mid x)$

$x \in \mathbb{R}^n$

parameters $w \in \mathbb{R}^n$, $b \in \mathbb{R}$

output: $\sigma(w^T x + b) = \hat{y}$

sigmoid function



$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

if $z \to +\infty$ $\sigma(z) \to 1$

if $z \to -\infty$ $\sigma(z) \to 0$

Cost function

Given $\{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\}$

we want to get $\hat{y}^{(i)} \approx y^{(i)}$

cross entropy loss

$$\mathcal{L}(\hat{y}, y) = -\left(y \log(\hat{y}) + (1-y) \log(1-\hat{y})\right)$$

if $y = 1 \Rightarrow \mathcal{L}(\hat{y}, y) = -\log(\hat{y})$ → we penalize the examples where $\hat{y} \approx 0$

$\Rightarrow$ we want $\hat{y}$ to be large $\Rightarrow$ we want $\hat{y}$ large

if $y = 0 \Rightarrow \mathcal{L}(\hat{y}, y) = -\log(1-\hat{y})$ → we penalize examples where $\hat{y} = 1$

$\Rightarrow$ we want $\hat{y}$ small

$\Rightarrow$ Cost:

$$J(w, b) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

$$J(w, b) = \frac{1}{m} \sum_{i=1}^{m} y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})$$

1. Gradient descent

Want to find w and b to minimize $J(w, b)$

$J(w, b)$ is convex in this case.

initialize w and b as w_0 and b_0

Repeat {

$$w := w - \alpha \cdot \frac{\partial J(w, b)}{\partial w}$$

$$b := b - \alpha \cdot \frac{\partial J(w, b)}{\partial b}$$

α: learning rate

}

for Logistic regression on a single training example

$z = w^T x + b$

$\hat{y} = \sigma(z)$

$\mathcal{L}(\hat{y}, y) = -(y \log(\hat{y}) + (1-y) \log(1-\hat{y}))$

$x_1, w_1, x_2, w_2, b \rightarrow z = x_1 w_1 + w_2 x_2 + b \rightarrow \hat{y} = \sigma(z) \rightarrow \mathcal{L}(a, y)$

$$\frac{d\mathcal{L}(\hat{y}, y)}{d\hat{y}} = -\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}}$$

$$\frac{d\mathcal{L}}{dz} = \frac{d\mathcal{L}}{d\hat{y}} \cdot \frac{d\hat{y}}{dz}, \quad \frac{d\hat{y}}{dz} = \hat{y}(1-\hat{y})$$

$$= \hat{y} - y$$

$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial z} \cdot \frac{\partial z}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial z} \cdot x_1$$

$$\frac{\partial \mathcal{L}}{\partial w_2} = \frac{\partial \mathcal{L}}{\partial z} \cdot \frac{\partial z}{\partial w_2} = \frac{\partial \mathcal{L}}{\partial z} \cdot x_2$$

$$\frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \mathcal{L}}{\partial z} \cdot \frac{\partial z}{\partial b} = \frac{\partial \mathcal{L}}{\partial z}$$

Gradient descent for m training examples

$$J(w,b) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}\left(\hat{y}^{(i)}, y^{(i)}\right)$$

$$\frac{\partial}{\partial w_1} J(w,b) = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial \mathcal{L}}{\partial w_1} \mathcal{L}\left(\hat{y}^{(i)}, y^{(i)}\right)$$

algorithm

$J=0 \quad dw_1 = 0 \quad dw_2 = 0 \quad db = 0$

for $i = 1$ to m

$$z^{(i)} = w^T x^{(i)} + b$$

$$a^{(i)} = \sigma(z^{(i)})$$

$$J_i = -\left[y^{(i)} \log\left(a^{(i)}\right) + (1-y^{(i)}) \log(1-a^{(i)})\right]$$

$$dz^{(i)} = \frac{dJ_i}{dz^{(i)}} = a^{(i)} - y^{(i)}$$

$$\begin{cases} dw_1 \mathrel{+}= x_1^{(i)} \cdot dz^{(i)} \\ dw_2 \mathrel{+}= x_2^{(i)} \cdot dz^{(i)} \\ db \mathrel{+}= dz^{(i)} \end{cases}$$

$J /= m$

$dw_1 /= m$

$dw_2 /= m$

$db /= m$

---

vectorizing this

$$w = \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix}$$

$dw = 0$

for $i = 1$ to m

$\vdots$

$$dw \mathrel{+}= x^{(i)} \cdot dz^{(i)}$$

$$db \mathrel{+}= x^{(i)} \cdot dz^{(i)}$$

---

vectorizing Logistic Regression

propagation steps

$$z^{(1)} = w^T x^{(1)} + b \quad ; \quad z^{(2)} = w^T x^{(2)} + b \quad | \quad \cdots$$
$$a^{(1)} = \sigma(z^{(1)}) \quad ; \quad a^{(2)} = \sigma(z^{(2)}) \quad |$$

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix} \in \mathbb{R}^{n \times m}$$

$$Z = \left[z^{(1)}, \ldots, z^{(m)}\right] \qquad W = \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix}$$

$$= w^T X + \underbrace{[b, \ldots, b]}_{(1,m)}$$

$$\boxed{Z = \text{np.dot}(w.T, X) + b}$$

$b \in \mathbb{R}$

calle operation fail

$$Z = \left[w^T x^{(1)} + b, \ldots, w^T x^{(m)} + b\right]$$

"broadcasting of b"

$$A = \left[a^{(1)}, \ldots, a^{(m)}\right] = \sigma(Z)$$

$$\boxed{A = \text{sigmoid}(Z)} \quad \text{(math.)}$$

$$dz^{(i)} = a^{(i)} - y^{(i)}$$

$$dZ = \left[dz^{(1)}, \ldots, dz^{(m)}\right]$$

$$\boxed{dZ = A - Y}$$

$$db = \frac{1}{m} \cdot \sum_{i=1}^{m} dz^{(i)}$$

$$\Rightarrow \boxed{db = \frac{1}{m} \cdot \text{np.sum}(dZ)}$$

$$dw = \frac{1}{m} \cdot X \cdot dZ^T$$

$$dw = 1/m \cdot \text{np.dot}(X, dZ.T)$$

W: a n by 1 vector
X: a n by m vector ; Y: a 1 by n vector

$J = 0$

$dW = 0$ # a n by 1 vector

$db = 0$

$Z = np.dot(W^T, X) + b$

$A = \sigma(Z)$

$dZ = A - Y$

$dw = 1/m \cdot X \cdot dZ^T$

$db = 1/m \cdot np.sum(dZ)$

$W = W - \alpha \cdot dW$

$b = b - \alpha \cdot db$

⇓

this is a single iteration of a gradient descent

$$\frac{\partial J}{\partial \theta_j} = \frac{\partial}{\partial \theta_j}\left[\frac{1}{m}\sum_{i=1}^{m} -y^{(i)}\log\left(h_\theta(x^{(i)})\right) - (1-y^{(i)})\log\left(1-h_\theta(x^{(i)})\right)\right]$$

$$= \frac{1}{m}\sum_{i=1}^{m} -y^{(i)}\frac{\partial}{\partial \theta_j}\log\left(h_\theta(x^{(i)})\right) - (1-y^{(i)})\frac{\partial}{\partial \theta_j}\log\left(1-h_\theta(x^{(i)})\right)$$

$z = \theta^T x$ $\quad u = \sigma(z)$ et $v = \log(u)$

pour un exemple x

$$\frac{\partial}{\partial \theta_j}\left(\log\left(h_\theta(x)\right)\right) = \frac{\partial}{\partial \theta_j}\left(\log\left(\sigma(\theta^T x)\right)\right)$$

$$= \frac{\partial v}{\partial u}\cdot\frac{\partial u}{\partial z}\cdot\frac{\partial z}{\partial \theta_j}$$

$$= \frac{1}{\sigma(z)}\,\sigma(z)\cdot(1-\sigma(z))\,x_j$$

$$= (1-h_\theta(x))\cdot x_j$$

et $\frac{\partial}{\partial \theta_j}\left(\log\left(1-h_\theta(x)\right)\right) = -h_\theta(x)\cdot x_j$

ainsi $\frac{\partial J}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x_j^{(i)}$

$$z = w_1 x_1 + w_2 x_2 + b$$

$$a = \frac{1}{1+e^{-z}}$$

$$\mathcal{L} = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log\left(a^{(i)}\right) + \left(1-y^{(i)}\right)\log\left(1-a^{(i)}\right)$$

$$w = w - \alpha \cdot \frac{\partial \mathcal{L}}{\partial w} \qquad b = b - \alpha \frac{\partial \mathcal{L}}{\partial b}$$

$\frac{\partial \mathcal{L}}{\partial w_1}$?

$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial a} \times \frac{\partial a}{\partial z} \times \frac{\partial z}{\partial w_1}$$

$$\frac{\partial \mathcal{L}}{\partial a} = -\frac{1}{m}\sum_{i=1}^{m} \frac{y^{(i)}}{a^{(i)}} - \frac{\left(1-y^{(i)}\right)}{1-a^{(i)}}$$

$$\frac{\partial a}{\partial z} = + \frac{e^{-z}}{(1+e^{-z})^2} = a(z)\left(1-a(z)\right)$$

$$\frac{\partial z}{\partial w_1} = x_1$$

$$\frac{\partial \mathcal{L}}{\partial w_1} = \left(-\frac{1}{m}\sum_{i=1}^{m} \frac{y^{(i)}}{a^{(i)}} - \frac{\left(1-y^{(i)}\right)}{1-a^{(i)}}\right) \times \left(a^{(i)}(z)\cdot\left(1-a^{(i)}(z)\right)\right) \times x_1$$

$$= -\frac{x_1}{m}\sum_{i=1}^{m} y^{(i)}\left(1-a^{(i)}\right) - \left(1-y^{(i)}\right)a^{(i)}$$

$$= -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)} - a^{(i)}\right) x_1^{(i)}$$

$$\frac{\partial \mathcal{L}}{\partial w_2} = -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)} - a^{(i)}\right) x_2^{(i)}$$

$$\frac{\partial \mathcal{L}}{\partial b} = -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)} - a^{(i)}\right)$$

$$X = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \cdots & x_n^{(1)} \\ \vdots & \vdots & & \vdots \\ x_1^{(m)} & x_2^{(m)} & \cdots & x_n^{(m)} \end{bmatrix} \in \mathbb{R}^{m \times n}$$

$$y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \\ y^{(m)} \end{bmatrix} \in \mathbb{R}^{m \times 1}$$

m : nombre de données

n : nombre de variables

$$W = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} \in \mathbb{R}^{n \times 1}$$

$$Z = X \cdot W + b$$

$$Z \in \mathbb{R}^{m \times 1}$$

$$A = \begin{bmatrix} \sigma(z_1) \\ \vdots \\ \sigma(z_n) \end{bmatrix} \in \mathbb{R}^{n \times 1} = \sigma(Z)$$

$$\mathcal{L} = -\frac{1}{m}\sum y \times \log(A) + (1-y)\times\log(1-A)$$

↓ produit terme à terme

vectorisation de la descente de gradient

$$W = \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix}$$

$$W = W - \alpha \cdot \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial w_1} \\ \vdots \\ \frac{\partial \mathcal{L}}{\partial w_n} \end{bmatrix}$$

$$b = b - \alpha \cdot \frac{\partial \mathcal{L}}{\partial b}$$

vectorisation des gradients

$$\frac{\partial \mathcal{L}}{\partial W} = \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial w_1} \\ \frac{\partial \mathcal{L}}{\partial w_2} \end{bmatrix} = \frac{1}{m} \cdot \begin{bmatrix} (a^{(1)} - y^{(1)}) x_1^{(1)} + (a^{(2)} - y^{(2)}) x_1^{(2)} + \dots + (a^{(m)} - y^{(m)}) x_1^{(m)} \\ (a^{(1)} - y^{(1)}) x_2^{(1)} + \dots + (a^{(m)} - y^{(m)}) x_2^{(m)} \end{bmatrix}$$

$$= \frac{1}{m} \cdot X^T \cdot (A - Y)$$

$$\frac{\partial \mathcal{L}}{\partial b} = \frac{1}{m} \sum_{i=1}^{m} (a^{(i)} - y^{(i)})$$

$$= \frac{1}{m} \cdot \sum A - Y$$

Un neurone formel

Définition 1

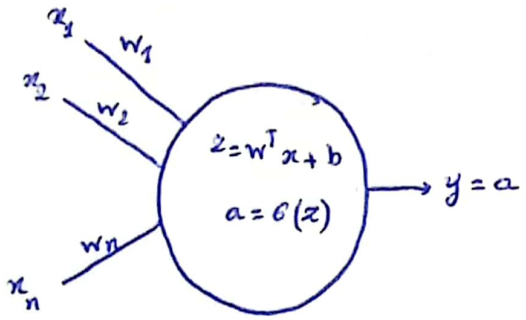
un neurone formel est une fonction paramétrée par $(n+1)$ paramètres $w_1, \dots, w_n, b$

$$g : \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R} \longrightarrow \mathbb{R}$$

$$(x, w, b) \longrightarrow g(x, w, b) = \sigma\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

σ : appelée fonction d'activation

chaque w_i : poids synaptique associé au signal d'entrée x_i



on a à notre disposition K points $x^{(k)} \in \mathbb{R}^n$ et $y^{(k)} \in \mathbb{R}$

Apprentissage du neurone

estimation par les moindres carrés des paramètres du neurone

Exemple 1

$$\sigma : \mathbb{R} \longrightarrow \mathbb{R}$$

$$x \longrightarrow \frac{1 - e^x}{1 + e^x}$$

$$r(\beta) : \mathbb{R}^{n+1} \longrightarrow \mathbb{R}^K$$

$$\beta \longrightarrow \begin{pmatrix} r_1(\beta) \\ \vdots \\ r_K(\beta) \end{pmatrix}$$

$\beta = \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \\ b \end{pmatrix}$ $\quad r_i(\beta) = y^{(i)} - g(x^{(i)}, w, b)$

$$(P) \begin{cases} \min \left(f(\beta) = \frac{1}{2} \|r(\beta)\|^2 \right) \\ \beta \in \mathbb{R}^{n+1} \end{cases}$$

2.2| ce n'est pas un problème linéaire

2.3| $\sigma \circ f : \mathbb{R} \longrightarrow \mathbb{R}$

$$x \longrightarrow \frac{1 - e^x}{1 + e^x}$$

$$r_i(\beta) = y^{(i)} - g(x^{(i)}, w, b)$$

$$= y^{(i)} - \sigma\left(\sum_{k=1}^{n} w_k x_k^{(i)} + b\right)$$

$$= y^{(i)} - b - \sum_{k=1}^{n} w_k x_k^{(i)}$$

$$r(\beta) = \begin{pmatrix} y^{(1)} \\ \vdots \\ y^{(K)} \end{pmatrix} - \begin{pmatrix} 1 & x_1^{(1)} & \cdots & x_n^{(1)} \\ \vdots & \vdots & & \vdots \\ 1 & x_1^{(K)} & & x_n^{(K)} \end{pmatrix} \beta$$

Le problème devient linéaire

Regression ~~Logistique~~ Logistique cas multiclasse

K classes avec $K > 2$

$\Rightarrow$ On utilise la fonction softmax

$$\mathbb{P}[y = i \mid x ; \theta] = \frac{e^{\theta_i x}}{\sum_{j=1}^{K} e^{\theta_j x}}$$

avec $y^{(i)} = {}^t(0 \dots 0 \, 1 \, 0 \dots 0)$: 1 est la i-ème coordonnée

$\Rightarrow$ Pour chaque vecteur de données de test $x^{(i)}$

On calcule un vecteur de probas

L'argmax de ces probabilités est la prédiction

$$h_\theta(x^{(i)}) = \begin{bmatrix} \mathbb{P}(y^{(i)} = 1 \mid x^{(i)} ; \theta) \\ \vdots \\ \mathbb{P}(y^{(i)} = K \mid x^{(i)} ; \theta) \end{bmatrix}$$

$$= \frac{1}{\sum_{j=1}^{K} e^{\theta_j x^{(i)}}} \begin{bmatrix} e^{\theta_1 x^{(i)}} \\ \vdots \\ e^{\theta_K x^{(i)}} \end{bmatrix}$$

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{K} \mathbb{1}\{y^{(i)} = j\} \log \left(\frac{e^{\theta_j x^{(i)}}}{\sum_{p=1}^{K} e^{\theta_p x^{(i)}}} \right)$$

$$\nabla_{\theta_j} J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} x^{(i)} \left(\mathbb{1}_{\{y^{(i)} = j\}} - \mathbb{P}(y^{(i)} = j \mid x^{(i)} ; \theta) \right)$$

exemple

classification : $y \in \{-1, 1\}$

$$\hat{y}(x) = f\left(\sum w_i x_i + b\right)$$

$$f(z) = \tanh(z) = \text{sigmoid}(2z) - \text{sigmoid}(-2z)$$

$$P(y, \hat{y}) = (y - \hat{y})^2$$

exemple 2

$$\hat{y}(w) = f\left(\sum_{i=1}^{n} w_i x_i\right)$$

f sigmoid

$$f(z) = \frac{1}{1 - e^{-z}}$$

$$J(w) = P(\hat{y}(w) \neq y)$$

1er cas : $P(\hat{y}, y) = -y \log(\hat{y}) - (1-y) \log(1-\hat{y})$

~~$J(w) = \sum_{i=1}^{m} P_i$~~

$$\nabla_{w_i} J(w) = P'(\hat{y}(w), y) \cdot \nabla_{w_i}(\hat{y}) \quad \text{(constante)}$$

$$= \left(-\frac{y}{\hat{y}(w)} - \frac{(1-y)}{1-\hat{y}(w)}\right) \cdot f'\left(\sum_{i=1}^{n} w_i x_i\right) \cdot x_i$$

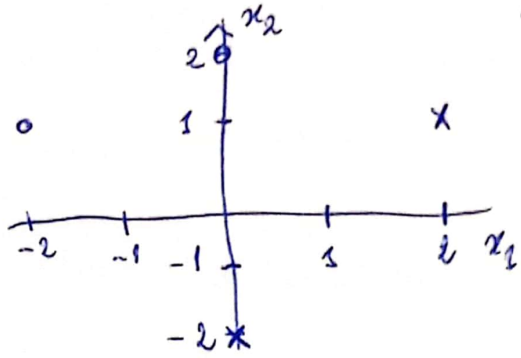
avec $f'(z) = \frac{e^{-z}}{(1-e^{-z})^2} = f'(z)\,(1 - f(z))$

2eme cas $P(\hat{y} \neq y) = (y - \hat{y})^2$: MSE

$$\nabla_{w_i} J(w) = P'(\hat{y}(w), y) \cdot \nabla_{w_i}(\hat{y})$$

$$= (\hat{y} - y) \cdot f'\left(\sum_{i=1}^{n} w_i x_i\right) \cdot x_i$$

x_1	x_2	y
2	1	1
0	-2	1
-2	1	0
0	2	0



$y = 1$
$0 = 0$

2/ pour que le perceptron sans biais peut separer les données avec la sigmoid activation

$$w = e\begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

$$\hat{y}^{(i)} = f\left(w^T x^{(i)}\right) \xrightarrow[\text{f} \to +\infty]{} y^{(i)}$$

$$J = \frac{1}{4} \sum_{i=1}^{4} P\left(\hat{y}^{(i)}, y^{(i)}\right) \xrightarrow[c \to +\infty]{} 0$$

avec $c = 1$

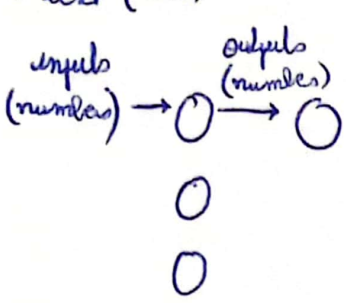
$$w^T x^{(i)} \begin{cases} +1 & \text{si } i = 1 \\ +2 & \text{si } i = 2 \\ -1 & \text{si } i = 3 \\ -2 & \text{si } i = 4 \end{cases}$$

Deep Learning

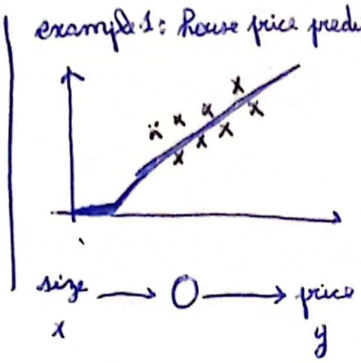
perceptrons multicouches

Neural Networks

* speech recognition
* images
* text (NLP)



example 1: house price prediction

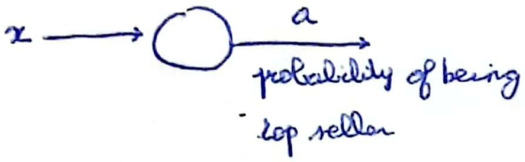


example :

x = price : input

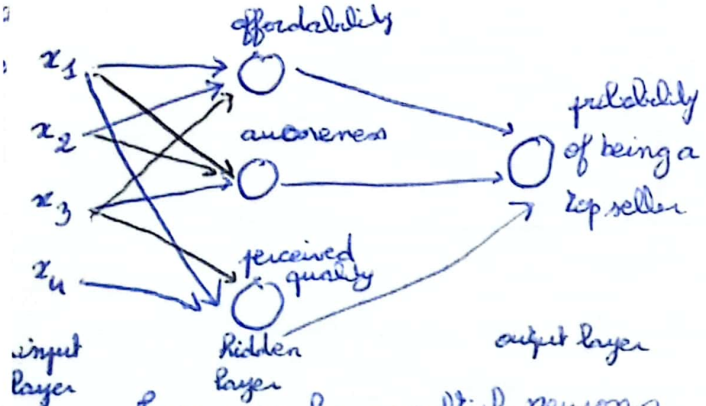
$a = f(x) = \frac{1}{1+e^{-(wx+b)}}$: output

activation



more complex example

x_1 : price
x_2 : shipping ~~cost~~
x_3 : marketing
x_4 : material



Layer : can have multiple neurons

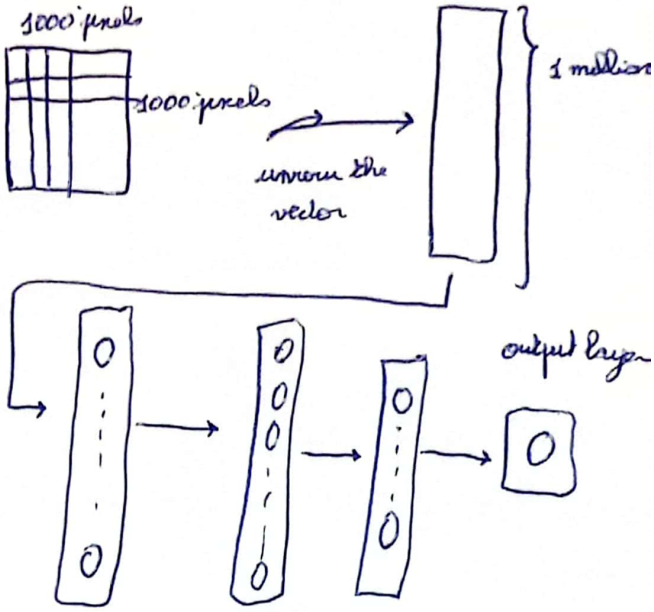
affordability
awareness
perceived quality
} activations

x_1
x_2
$x_3 = x_4$
} input layer

in practice

every neuron will have access to every feature from the previous layer

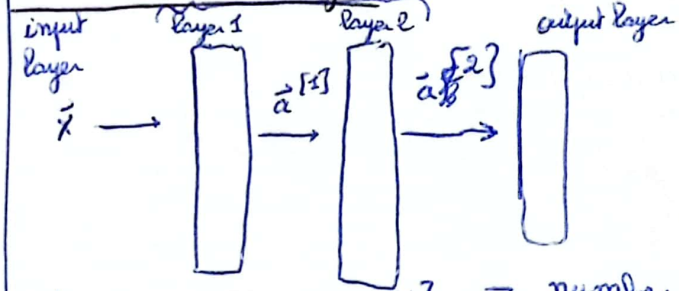
example : image recognition



~~neurons~~

first layer : neurons are looking for short lines
second layer : group together short lines in order to look for parts of faces

neural network layer → hidden layers



the dimension of $a^{[i]}$ = number of neurons in layer i

~~depends~~

profondeur du réseau = 3 : 2 hidden layers + 1 output layer

binary choice in output layer :

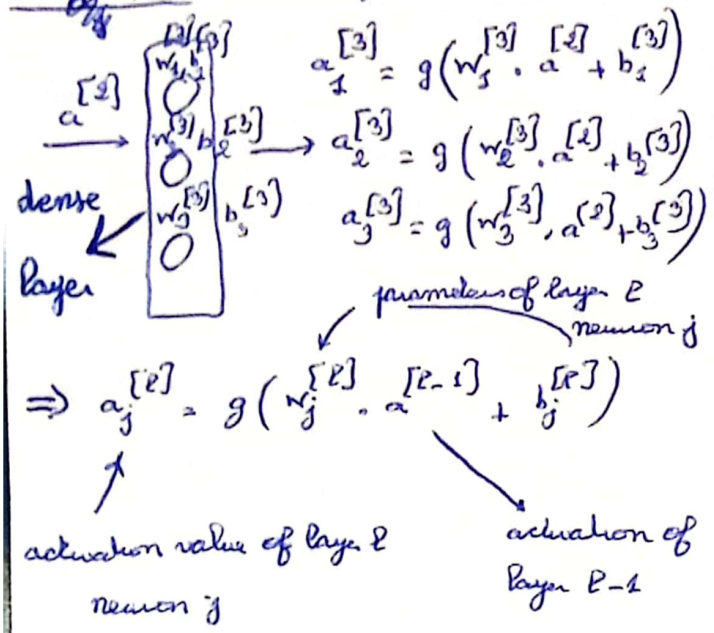
we need one neuron

more complex neural network

4 layers : 3 hidden layers + output layer

⇒ we don't count the input layer

example : activation function = sigmoid



NB :

$\vec{x} \rightarrow \vec{a}_1 \rightarrow \vec{a}_2 \rightarrow \vec{a}_3$:

forward propagation

Exemple basique :

regression linéaire simple = un seul neurone + sans activation

---

étapes

1/ forward propagation : de la couche d'entrée à la couche de sortie

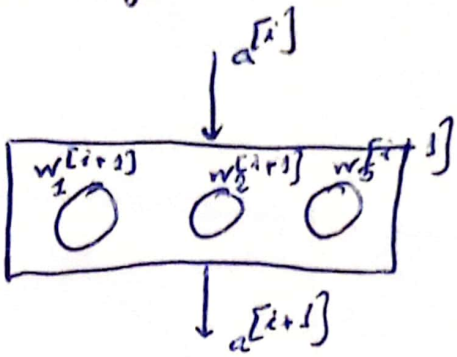
2/ calcul de l'erreur de sortie après la propagation des données

3/ calcul des gradients d'erreurs pour la couche de sortie ⇒ corriger les poids synaptiques de la couche de sortie

4/ calcul des gradients d'erreur pour corriger les poids synaptiques des neurones des couches cachés

5/ mise à jour des poids synaptiques

1/ dense layer

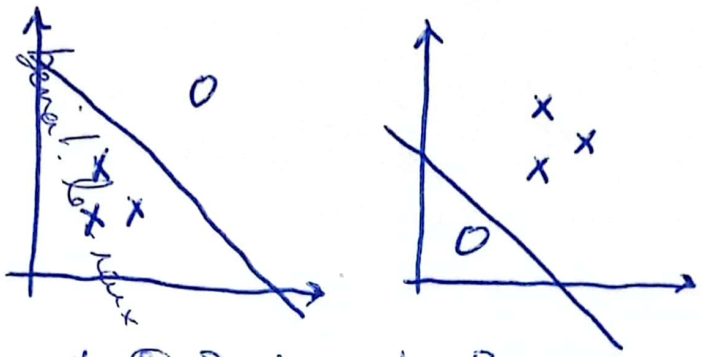


1/ multiplier par le poids

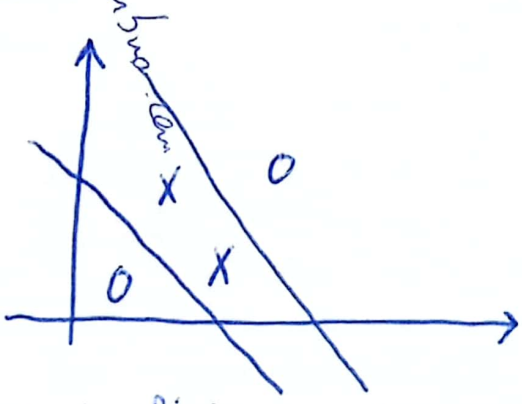
2/ ajouter le biais

3/ fonction d'activation

Remarque : un neurone crée un seul barrière de décision



cette classification peut se faire avec un seul neurone



cette classification ne peut pas se faire avec un seul neurone en dense layer

Mais doit se faire en deux

Dans ce cas on a besoin :

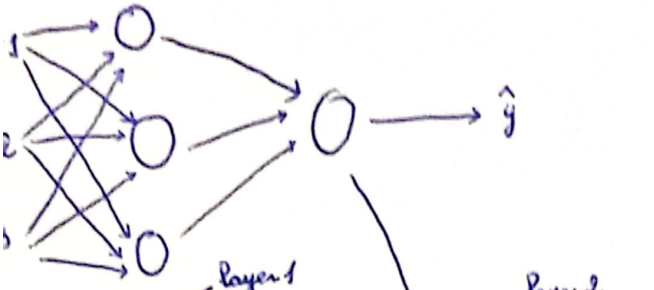
1 dense layer with 2 nodes

1 dense layer with 1 node

$z^{[1]} = Wx + b^{[1]}$

Neural networks

I| only one example



Layer 1 → $W^{[1]}$, $b^{[1]}$ ← Layer 1

Layer 2

$$z^{[1]} = W^{[1]} x + b^{[1]}$$
$$a^{[1]} = \sigma(z^{[1]})$$

$$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$$
$$a^{[2]} = \sigma(z^{[2]})$$

---

parameters of Layer 1: $W^{[1]}$, $b^{[1]}$

parameters of Layer 2: $W^{[2]}$, $b^{[2]}$

in our case:

in Layer 1: we have 4 neurons + 3 inputs

$\Rightarrow W^{[1]}.shape = (4,3)$

$b^{[1]}.shape = (4,1)$;

in Layer 2: we have 1 neuron 4 inputs

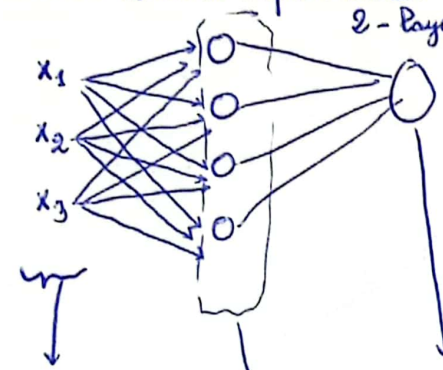
$W^{[2]}.shape = (1,4)$

$b^{[2]}.shape = (1,1)$

---

neural network representation

2 - Layer neuron network



⚠ we don't count the input layer in the number of layers

input layer $X^{(i)}$

hidden layer $a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{bmatrix}$

output layer $\hat{y} = a^{[2]}$

a single example

$a^{[0]} = X$

ivation

ue

the dimension of $a^{[i]}$ = number of neurons in layer i

the dimension of $a^{[0]}$ = number features

---

what does this NN compute

$X = \begin{bmatrix} | & | & | \\ x^{(1)} & x^{(2)} & x^{(3)} \\ | & | & | \end{bmatrix}$ $\quad x^{(i)} = \begin{bmatrix} x_1^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix}$ the i^{th} example

First Layer

$$z_1^{[1]} = w_1^{[1]T} \cdot X + b_1^{[1]}$$

first neuron

$$a_1^{[1]} = \sigma(z_1^{[1]})$$

dimensions

$w_1^{[1]} : (4,1)$

$X : (n,1)$

$b_1^{[1]} : (1,1)$

$z_1^{[1]} : (1,1)$

$a_1^{[1]} : (1,1)$

$a_i^{[l]} = w_i^{[l]} \cdot a^{[l]} + b_i^{[l]}$ ← Layer l

node i

$$z_1^{[1]} = w_1^{[1]T} X + b_1^{[1]}$$
$$z_2^{[1]} = w_2^{[1]T} X + b_2^{[1]}$$
$$z_3^{[1]} = w_3^{[1]T} X + b_3^{[1]}$$
$$z_4^{[1]} = w_4^{[1]T} + b_3^{[1]}$$

$\Leftrightarrow$

$$W^{[1]} = \begin{bmatrix} - & w_1^{[1]T} & - \\ - & w_2^{[1]T} & - \\ - & w_3^{[1]T} & - \\ - & w_4^{[1]T} & - \end{bmatrix}$$

$$Z^{[1]} = \begin{bmatrix} z_1^{[1]} \\ \vdots \\ z_4^{[1]} \end{bmatrix}$$

$$z^{[1]}_{(4,1)} = W^{[1]}_{(4,3)} x_{(3,1)} + b^{[1]}_{(4,1)}$$
$$a^{[1]}_{(4,1)} = \sigma(z^{[1]}_{(4,1)})$$
$$z^{[2]}_{(1,1)} = W^{[2]}_{(1,4)} a^{[1]}_{(4,1)} + b^{[2]}_{(1,1)}$$
$$a^{[2]}_{(1,1)} = \sigma(z^{[2]}_{(1,1)})$$

dimensions

x : $(n, 1)$

$z^{[1]}$: $(n^{[1]}, 1)$

$W^{[1]}$: $(n^{[1]}, n)$

$b^{[1]}$: $(n^{[1]}, 1)$

$a^{[1]}$: $(n^{[1]}, 1)$

$W^{[2]}$: $(n^{[2]}, n^{[1]})$

$b^{[2]}$: $(n^{[2]}, 1)$

$z^{[2]}$: $(n^{[2]}, 1)$

$a^{[1]}$: $(n^{[2]}, 1)$

⚠ The dimension of $a^{[i]}$ = number of neurons in layer i

II - accross multiple training examples

we have m training examples

$x^{(1)}, \dots, x^{(m)}$

$x^{(1)} \longrightarrow a^{[2](1)} = \hat{y}^{(1)}$

$\vdots$

$x^{(m)} \longrightarrow a^{[2](m)} = \hat{y}^{(m)}$

for i = 1 to m # loop over m training examples

$z^{[1](i)} = W^{[1]} x^{(i)} + b^{[1]}$

$a^{[1](i)} = \sigma(z^{[1](i)})$

$z^{[2](i)} = W^{[2]} a^{[1](i)} + b^{[2]}$

$a^{[2](i)} = \sigma(z^{[2](i)})$

Vectorization

$$X = \begin{bmatrix} | & & | \\ x^{(1)} & \cdots & x^{(m)} \\ | & & | \end{bmatrix} \in \mathcal{M}_{n,m}(\mathbb{R})$$

from $z^{[1]} \longrightarrow Z^{[1]} = \begin{bmatrix} | & & | \\ z^{[1](1)} & z^{[1](2)} & \cdots & z^{[1](m)} \\ | & & | \end{bmatrix}$

$Z^{[1]} = W^{[1]} X + b^{[1]}$

$A^{[1]} = \sigma(Z^{[1]})$

$Z^{[2]} = W^{[2]} X + b^{[2]}$

$A^{[2]} = \sigma(Z^{[2]})$

pour tous les exemples

$W^{[1]}, W^{[2]}, b^{[1]}, b^{[2]}$ ne varient pas selon l'indice

~~$w^{[1](1)}$~~

dimensions

$X : (n, m)$

$W^{[1]} : (n^{[1]}, n)$

$A^{[1]}$ et $Z^{[1]} : (n^{[1]}, m)$

$W^{[2]} : (n^{[2]}, n^{[1]})$

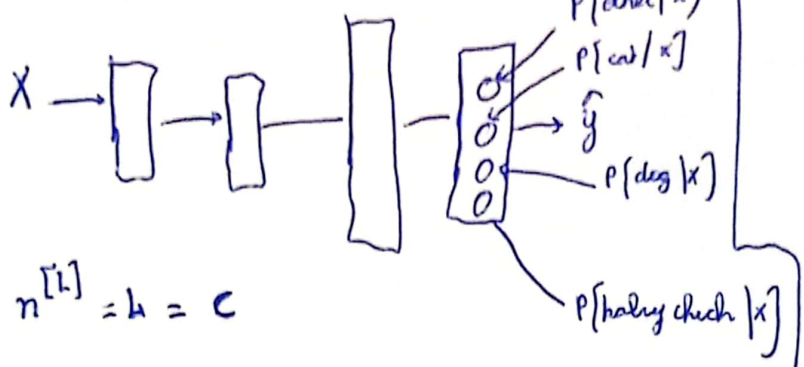
$A^{[2]}$ et $Z^{[2]} : (n^{[2]}, m)$

Softmax Regression for multiclassification

example: cats VS dogs VS baby chicks VS other

1 2 3 0

C = # classes = 4



$n^{[L]} = 4 = C$

$\hat{y}$ is $(4,1)$ $\hat{y}$ is $(C,1)$

$z^{[L]} = w^{[L]} a^{[L-1]} + b^{[L]}$

$t = e^{(z^{[L]})}$

$$a^{[L]} = \frac{e^{(z^{[L]})}}{\sum_{i=1}^{C} t_i}$$

⚠ y must be also $(C,1)$

if we have a cat $y = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$ $\hat{y} = \begin{bmatrix} 0.3 \\ 0.2 \\ 0.1 \\ 0.4 \end{bmatrix}$

$$\mathcal{L}(\hat{y}, y) = -\sum_{j=1}^{4} y_j \log(\hat{y}_j)$$

$$= -y_2 \log(\hat{y}_2)$$

$$J(w^{[1]}, b^{[1]}, \dots) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

$$Y = \left[y^{(1)}\ y^{(2)}, \dots, y^{(m)} \right]$$

$$= \begin{bmatrix} 0 & 0 & 1 & \dots \\ 1 & 0 & 0 & \\ 0 & 1 & 0 & \\ 0 & 0 & 0 & \end{bmatrix}$$

$(4, m)$

⚠ if we have

$y = [1, 2, 0, \dots]$

on veut

$$Y = \begin{bmatrix} 0 & 0 & 1 & \dots \\ 1 & 0 & 0 & \\ 0 & 1 & 0 & \\ 0 & 0 & 0 & \end{bmatrix}$$

we use one hot encodings

in TensorFlow

y_one_hot = tf.reshape(tf.one_hot(Label, C, axis=0), (depth,))

ceci est appliqué à un seul exemple

if we have our labels in this form

$$Y = \begin{bmatrix} | & | & & | \\ y^{(1)} & y^{(2)} & \cdots & y^{(m)} \\ | & | & & | \end{bmatrix} \quad (C, m)$$

$$\hat{Y} = \begin{bmatrix} | & | & & | \\ \hat{y}^{(1)} & \hat{y}^{(2)} & \cdots & \hat{y}^{(m)} \\ | & | & & | \end{bmatrix}$$

en tensorflow :

to compute categorical cross entropy :

- about the shape :

$$Y = \begin{bmatrix} \text{---} & y^{(1)} & \text{---} \\ \text{---} & y^{(2)} & \text{---} \\ & \vdots & \\ \text{---} & y^{(m)} & \text{---} \end{bmatrix} \quad (m, C)$$

- about logits

if in the forward propagation

1/ we finished with $Z^{[L]}$ without calculating $A^{[L]}$

⇒ we need to set from_logits = True in the parameters of the cross entropy

2/ we finished by $A^{[L]}$

⇒ set from_logits = False

Multiclass classification with neural network

① Logistic Regression
(2 possible output values)

$$z = \vec{w} \cdot \vec{x} + b$$

$$a_1 = g(z) = \frac{1}{1+e^{-z}} = P(y=1 \mid \vec{x})$$

$$a_2 = 1 - a_1 = P(y=0 \mid \vec{x})$$

② softmax regression

$y = 1, 2, 3, 4$

$$z_1 = \vec{w}_1 \cdot \vec{x} + b_1$$

$$a_1 = \frac{e^{z_1}}{\underbrace{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}}_{\Delta}} = P(y=1 \mid \vec{x})$$

$$z_2 = \vec{w}_2 \cdot \vec{x} + b_2$$

$$a_2 = \frac{e^{z_2}}{\Delta} = P(y=2 \mid \vec{x})$$

$$z_3 = \vec{w}_3 \cdot \vec{x} + b_3$$

$$a_3 = \frac{e^{z_3}}{\Delta} = P(y=3 \mid \vec{x})$$

$$z_4 = \vec{w}_4 \cdot \vec{x} + b_3$$

$$a_4 = \frac{e^{z_4}}{\Delta} = P(y=4 \mid \vec{x})$$

③ softmax Regression generalization

$$z_j = \vec{w}_j \cdot \vec{x} + b_j \quad j = 1, \dots, N$$

$$a_j = \frac{e^{z_j}}{\sum_{k=1}^{N} e^{z_k}} = P(y=j \mid \vec{x})$$

$$\sum_{i=1}^{N} a_i = 1$$

cost function

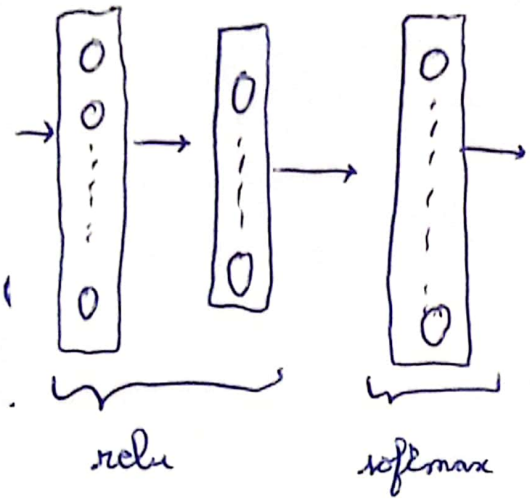
$$\text{Loss} = -y \log(a_1) - (1-y) \log(a_2)$$

$J(\vec{w}, b)$ = average loss

Cost

$$\text{Loss}(a_1, \dots, a_N, y) = \begin{cases} -\log a_1 & \text{if } y=1 \\ -\log a_2 & \text{if } y=2 \\ \vdots & \\ -\log a_N & \text{if } y=N \end{cases}$$

softmax with NN



Logistic Regression

$a_1^{[3]} = g(z_1^{[3]})$ et $a_2^{[3]} = g(z_2^{[3]})$

Softmax Regression

$a^{[3]} = (a_1^{[3]}, \dots, a_{10}^{[3]}) = g(z_1^{[3]}, \dots, z_{10}^{[3]})$

```
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
model = Sequential([
    Dense(units = 25, activation = 'relu'),
    Dense(units = 15, activation = 'relu'),
    Dense(units = 10, activation = 'softmax'),
])
```

```
from tensorflow.keras.losses import
    SparseCategoricalCrossentropy
model.compile(loss = SparseCategoricalCrossentropy
    ())
```

```
model.fit(X, y, epochs = 100)
```

⇒ the model output is $a_1, \dots \to a_{10}$

improved version SparseCategorical

```
① model.compile(loss = ~~Binary~~CrossEntropy(
        from_logits = True))
```

② set the output layer activation function to 'linear'

Logits = z

numerical Roundoff errors

$x = \frac{2.0}{10000} \to 0,00020000$

$x_2 = 1 + (1/10000) - (1 - 1/10000)$

$\to 0,000199999\ldots$

improved implementation of Logistic Loss

① on calcule a et puis on calcule la loss en fonction de a

$a = \frac{1}{1+e^{-z}}$

$loss = -y \log(a) - (1-y) \log(1-a)$

② improved version : ne pas calculer a en intermediaire

$loss = -y \log\left(\frac{1}{1+e^{-z}}\right) - (1-y) \log\left(1 - \frac{1}{1+e^{-z}}\right)$