

RÉSEAU DE NEURONES DENSE

1 Régression logistique en dimension 2



Point Méthode : Régression logistique à 2 classes

Étant donné un jeu de données de points X et de classes T , un *learning rate* lr et un nombre d'itérations nb_iter .

1. **Initialisation des paramètres** : on initialise (W, b) aléatoirement.
2. **Boucle** : itérer un nombre nb_iter de fois :
 - **Prédiction** : $Y = \sigma(XW + b)$.
 - **Optimisation** : $W = W - lr \cdot \nabla_W J(Y) = W - lr \cdot {}^t X(Y - T)$.
 - **Optimisation** : $b = b - lr \cdot \nabla_b J(Y) = b - lr \cdot \text{np.sum}(Y - T)$.
3. **Résultat** : On renvoie les paramètres optimaux (W, b) .

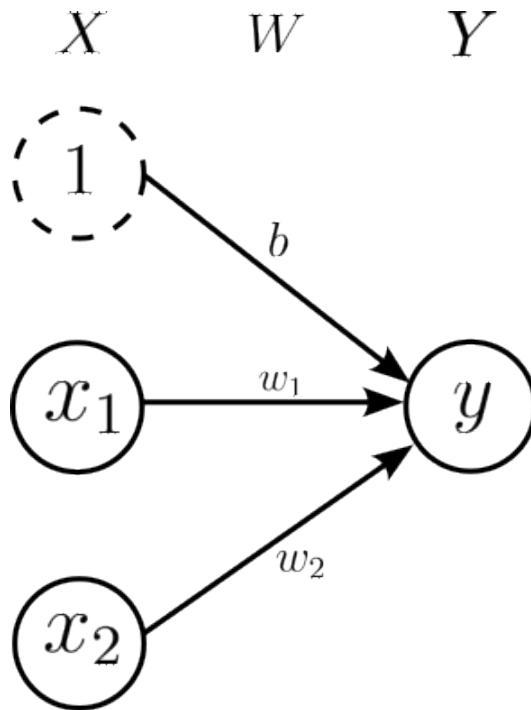


FIGURE 1 – Schéma de la régression logistique

2 Réseau de neurones dense

Définition 2.1 (*Opérateur \odot*)

L'opérateur \odot est défini pour deux matrices A, B de même taille (n, k) comme la matrice de taille (n, k) :

$$(A \odot B)_{i,j} = a_{i,j} b_{i,j}$$

On dit également que c'est le **produit termes à termes**.

En python, l'opération $A * B$ calcule $A \odot B$ pour deux `np.array`.

2.1 Réseau dense à 1 couche intermédiaire

Définition 2.2 (*Réseau de neurones dense à 1 couche intermédiaire*)

Les éléments composant le réseau de neurones de gauche à droite sont :

- Des **données** X de taille (N, D) .
- Des **paramètres** W_1 de taille (D, C) et $b_1 \in \mathbb{R}$.
- Des **données intermédiaires** Z de taille (N, C) .
- Des **paramètres** W_2 de taille $(C, 1)$ et $b_2 \in \mathbb{R}$.
- Des **résultats** Y de taille $(N, 1)$.

L'étape de **prédition** est faite de gauche à droite :

- $Z = \sigma(X \cdot W_1 + b_1)$
- $Y = \sigma(Z \cdot W_2 + b_2)$

L'étape d'**optimisation par rétropagation** (back-propagation) est faite de droite à gauche :

- $\delta_2 = Y - T$
- $\nabla_{W_2} J = {}^t Z \cdot \delta_2$
- $\nabla_{b_2} J = \text{delta2.sum}()$.
- $\delta_1 = (\delta_2 \cdot {}^t W_2) \odot Z \odot (1 - Z)$.
- $\nabla_{W_1} J = {}^t X \cdot \delta_1$
- $\nabla_{b_1} J = \text{delta1.sum}()$.

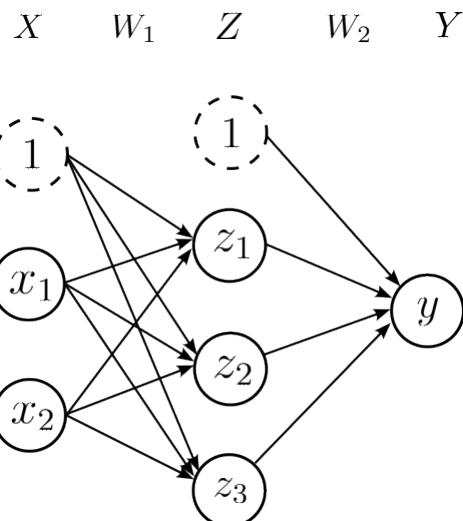


FIGURE 2 – Schéma d'un réseau à 1 couche intermédiaire

Remarque 1

On relie tous les x_i à tous les z_j car on ne sait pas ce que représente la variable z_j et donc comment elle utilise les variables x_i .

2.2 Réseau dense à 2 couches intermédiaires

Définition 2.3 (*Réseau de neurones dense à 2 couches intermédiaires*)

Les éléments composant le réseau de neurones de gauche à droite sont :

- Des **données** X de taille (N, D) .
- Des **paramètres** W_1 de taille (D, C_1) et $b_1 \in \mathbb{R}$.
- Des **données intermédiaires** Z_1 de taille (N, C_1) .
- Des **paramètres** W_2 de taille (C_1, C_2) et $b_2 \in \mathbb{R}$
- Des **données intermédiaires** Z_2 de taille (N, C_2) .
- Des **paramètres** W_3 de taille $(C_2, 1)$ et $b_3 \in \mathbb{R}$
- Des **résultats** Y de taille $(N, 1)$.

L'étape de **prédition** est faite de gauche à droite :

- $Z_1 = \sigma(X \cdot W_1 + b_1)$
- $Z_2 = \sigma(Z_1 \cdot W_2 + b_2)$
- $Y = \sigma(Z_2 \cdot W_3 + b_3)$

L'étape d'**optimisation** par back-propagation est faite de droite à gauche :

- $\delta_3 = Y - T$
- $\nabla_{W_3} J = {}^t Z_2 \cdot \delta_3$
- $\nabla_{b_3} J = \text{delta3.sum}()$.
- $\delta_2 = (\delta_3 \cdot {}^t W_3) \odot Z_2 \odot (1 - Z_2)$.
- $\nabla_{W_2} J = {}^t Z_1 \cdot \delta_2$
- $\nabla_{b_2} J = \text{delta2.sum}()$.
- $\delta_1 = (\delta_2 \cdot {}^t W_2) \odot Z_1 \odot (1 - Z_1)$.
- $\nabla_{W_1} J = {}^t X \cdot \delta_1$
- $\nabla_{b_1} J = \text{delta1.sum}()$.

2.3 Réseau dense à p couches intermédiaires

Définition 2.4 (*Réseau de neurones dense à p couches intermédiaires*)

- Des **données** X de taille (N, D) .
- Des **paramètres** W_1 de taille (D, C_1) et $b_1 \in \mathbb{R}$.
- Des **données intermédiaires** Z_1 de taille (N, C_1) .
- Pour $i \in \llbracket 2, p \rrbracket$, des **paramètres** W_i de taille (C_{i-1}, C_i) et $b_i \in \mathbb{R}$.
- Pour $i \in \llbracket 2, p \rrbracket$, des **données intermédiaires** Z_i de taille (N, C_i)
- Des **paramètres** W_{p+1} de taille $(C_p, 1)$ et $b_{p+1} \in \mathbb{R}$.
- Des **résultats** Y de taille $(N, 1)$.

La **prédition** est faite de gauche à droite :

- $Z_1 = \sigma(X \cdot W_1)$
- $\forall i \in \llbracket 2, p \rrbracket, Z_i = \sigma(Z_{i-1} \cdot W_i) \quad (i \text{ croissants})$
- $Y = \sigma(Z_p \cdot W_{p+1})$

L'étape d'**optimisation** par back-propagation est faite de droite à gauche :

- $\delta_{p+1} = Y - T$.
- $\forall i \in \llbracket 1, p \rrbracket, \begin{cases} \nabla_{W_{i+1}} J = {}^t Z_i \cdot \delta_{i+1} \\ \nabla_{b_{i+1}} J = \text{delta}_{i+1}.sum() \\ \delta_i = (\delta_{i+1} \cdot {}^t W_{i+1}) \odot Z_i \odot (1 - Z_i) \end{cases} \quad (i \text{ décroissants})$
- $\nabla_{W_1} J = {}^t X \cdot \delta_1$
- $\nabla_{b_1} J = \text{delta1.sum}()$.

3 Régression Logistique à $K \geq 3$ classes

Dans cette section on définit la régression logistique à $K \geq 3$ classes. La différence majeure est que les matrices T et Y seront de taille (N, K) et que l'on remplace la fonction sigmoïde σ par la fonction *softmax*. Chaque calcul de prédiction à une classe est effectué séparément pour chacune des classes.

Définition 3.1 (*Matrice de classes T*)

*La matrice T donnant les classes réelles est une matrice de 0 et de 1 ayant **exactement un 1 par ligne**.*

$$T = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

On appelle cette approche le one hot encoding.

Définition 3.2 (*Fonction softmax*)

On définit la fonction softmax du n-uplets (x_1, x_2, \dots, x_n) comme :

$$\text{softmax}(x_1, x_2, \dots, x_n) = \frac{1}{\sum_{k=1}^K e^{x_k}} (e^{x_1}, e^{x_2}, \dots, e^{x_K})$$

Définition 3.3 (*Fonction erreur de cross entropy*)

On définit la fonction d'erreur J par :

$$J = - \sum_{n=1}^N \sum_{k=1}^K t_{nk} \ln(y_{nk})$$

Définition 3.4 (*Régression logistique à $K \geq 3$ classes*)

- Des données X de taille (N, D) .
- Des paramètres W de taille (D, K) et une matrice de biais b de taille $(1, K)$.
- Des résultats Y de taille (N, K) .

L'étape de prédiction est donnée par :

- $Y = \text{softmax}(XW + b)$

L'étape de est donnée par :

- La fonction d'erreur à optimiser est :

$$J = \sum_{n=1}^N \sum_{k=1}^K t_{nk} \ln(y_{nk})$$

- $W = W - lr * {}^t X \cdot (Y - T)$
- $b = b - lr \cdot \nabla_b J(Y) = b - lr \cdot np.sum(Y - T, axis = 0)$.

Exercice 1

1. Pourquoi impose-t-on que T possède exactement un 1 par ligne ?
2. Justifier que la fonction *softmax* est à valeurs de $[0, 1]$ et que la somme de tous ses termes vaut 1.
3. Expliquer le choix de cette fonction d'erreur d'entropie. On expliquera les comportements de (t_{nk}, y_{nk}) qui la rendent faible ou élevée.
4. Proposer une généralisation des réseaux de neurones denses aux cas d'un problème de classification de données X réparties en $K \geq 3$ classes.

**Point Méthode : Régression logistique à K classes**

Étant donné un jeu de données de points X et de classes T , un *learning rate* `lr` et un nombre d'itérations `nb_iter`.

1. **Initialisation des paramètres** : on initialise (W, b) aléatoirement.
2. **Boucle** : itérer un nombre `nb_iter` de fois :
 - **Prédiction** : $Y = softmax(XW + b)$.
 - **Optimisation** : $W = W - lr \cdot \nabla_W J(Y) = W - lr \cdot {}^t X(Y - T)$.
 - **Optimisation** : $b = b - lr \cdot \nabla_b J(Y) = b - lr \cdot np.sum(Y - T, axis = 0)$.
3. **Résultat** : On renvoie les paramètres optimaux (W, b) .