Workshop Build a Brain

IALab, Quentin Torroba

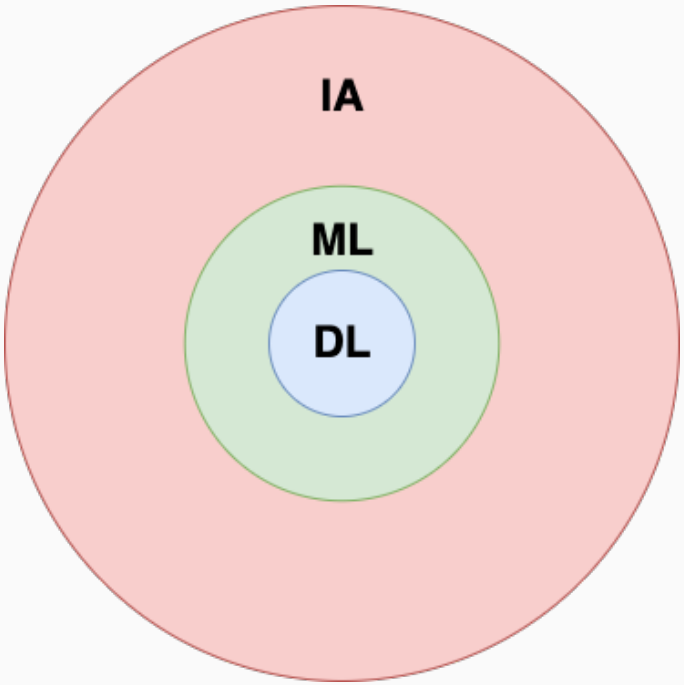
15 Novembre 2023

Tout d'abord

allez sur https://github.com/Garage-ISEP/build-a-brain

Introduction au Deep Learning

Concept 1: Le Deep Learning



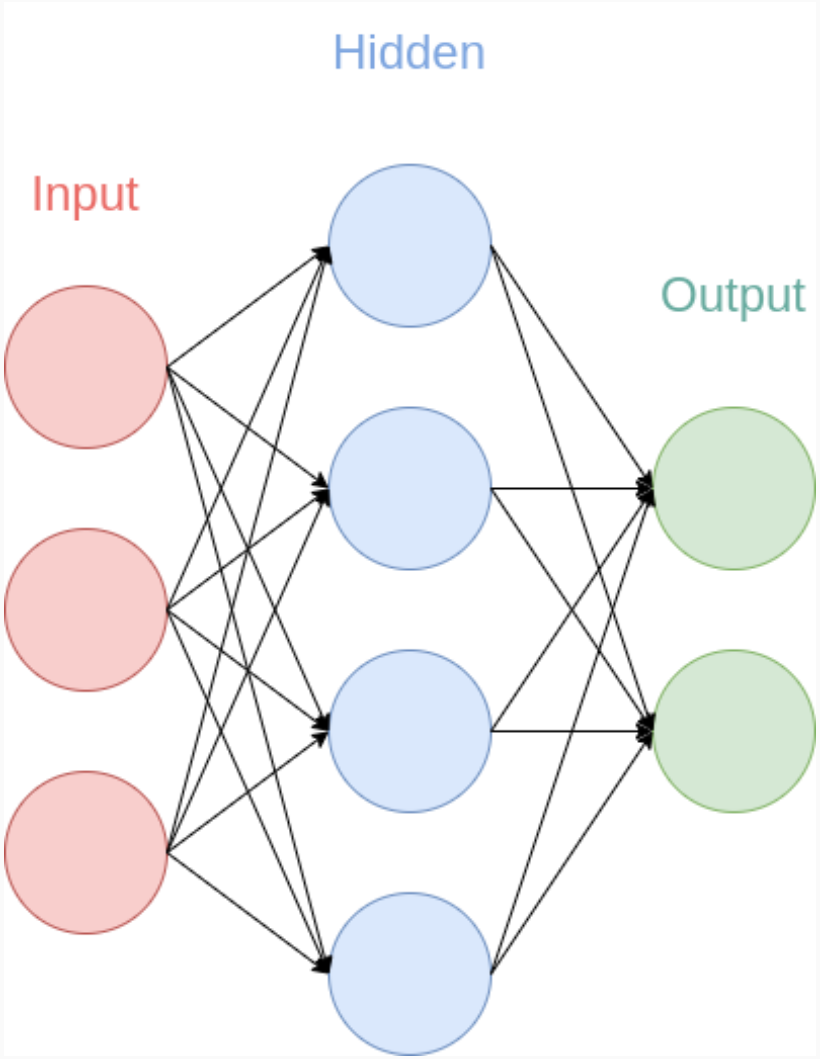
- **Intelligence Artificielle (IA)**: Reproduction de l'intelligence humaine via des algorithmes. Exemple: Algorithme pour battre Sudoku.
- **Machine Learning**: Utilisation de méthodes statistiques pour apprendre à partir des données. Exemple: kNN
- **Deep Learning**: Systèmes apprenant d'eux-mêmes à partir de données, principalement via des réseaux de neurones.

Concept 2: Le “Bas niveau” du Deep Learning

- **Concepts en informatique**:
  - **Bas niveau**: Langages comme l’assembleur, proches de la machine.
  - **Haut niveau**: Langages comme Python, abstraits et faciles à utiliser.
- **Parallèle en Deep Learning**:
  - **Haut niveau**: Bibliothèques comme PyTorch cachant la complexité.
  - **Bas niveau**: Mathématiques et mécanismes fondamentaux des réseaux de neurones.
- **Focus d’aujourd’hui**: Comprendre les bases et les mathématiques du deep learning.

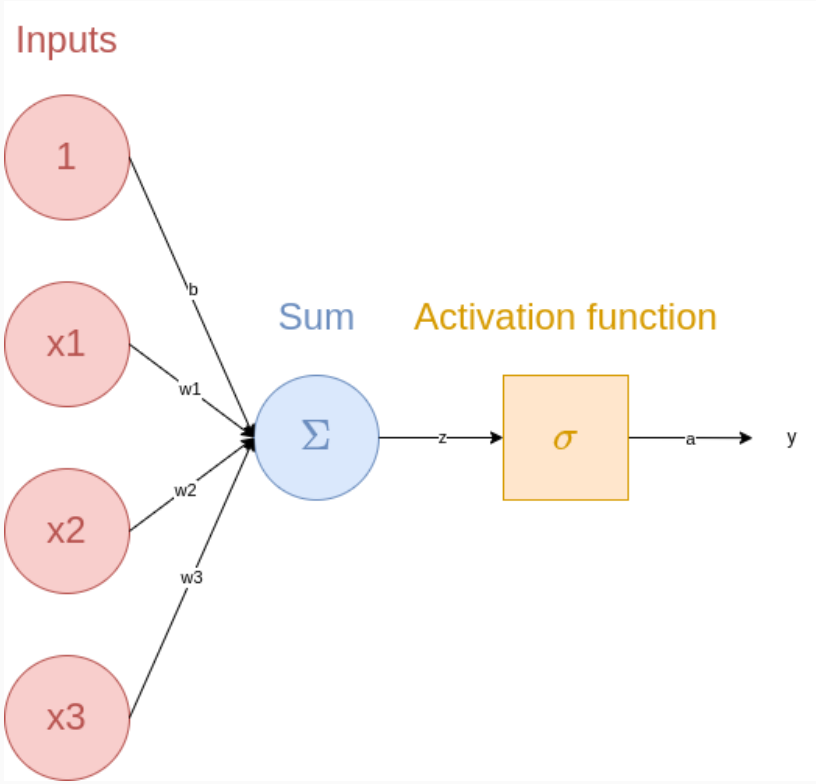
Introduction au Perceptron

Réseaux de Neurones



- **Couche d'entrée (Input Layer)**: Représente les données d'entrée (Ex: Pixels d'une image)
- **Couche de sortie (Output Layer)**: Représente les données de sortie (Ex: Probabilité d'être un chat)
- **Couches cachées (Hidden Layers)**: Composées d'éléments appelés perceptrons

Fonctionnement



1. **Poids et Biais**: Importance relative de chaque entrée, ajustement du point de départ
2. **Combinaison linéaire**: Multiplication des entrées pondérées et addition du biais
3. **Fonction d'activation**: Introduit la complexité et la non-linéarité

Exemple: Approximation Linéaire

Objectif

- Approximer $f(x) = 2x + 1$ avec un perceptron
- Montrer qu'un perceptron permet une régression linéaire

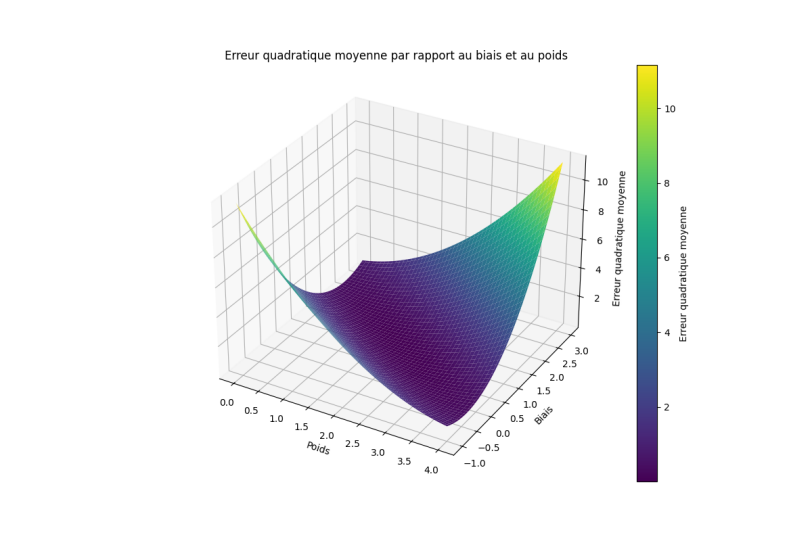
Architecture du perceptron

- Création du poids et du biais
- Pas de fonction d'activation
- Sortie du perceptron: $z = w \cdot x + b$

Implémentation en Python

Choix de la Fonction de Perte

Fonction de perte



- **Nécessaire pour quantifier et corriger l'erreur**
- **Erreur Quadratique Moyenne (MSE):** Couramment utilisée en régression

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

- Fonction de coût: $\mathcal{C} = \frac{MSE}{2}$

Descente de Gradient et Rétropropagation

Concept

- Minimiser l'erreur en suivant la direction opposée du gradient
- **Calcul du gradient**: Règle de chaîne
- **Ajustement des paramètres (poids et biais)**

Calcul des gradients

$$\frac{\partial \mathcal{C}}{\partial w} = \frac{\partial \mathcal{C}}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w}$$

et

$$\frac{\partial \mathcal{C}}{\partial b} = \frac{\partial \mathcal{C}}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial b}$$

Sachant que $\mathcal{C} = \frac{1}{2n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$ et $\hat{y} = wx + b$ avec x l'entrée, on a alors

$$\frac{\partial \mathcal{C}}{\partial w} = \frac{1}{2n} 2 \sum_{i=1}^{n} (y_i - \hat{y}_i) \cdot x = \frac{x}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)$$

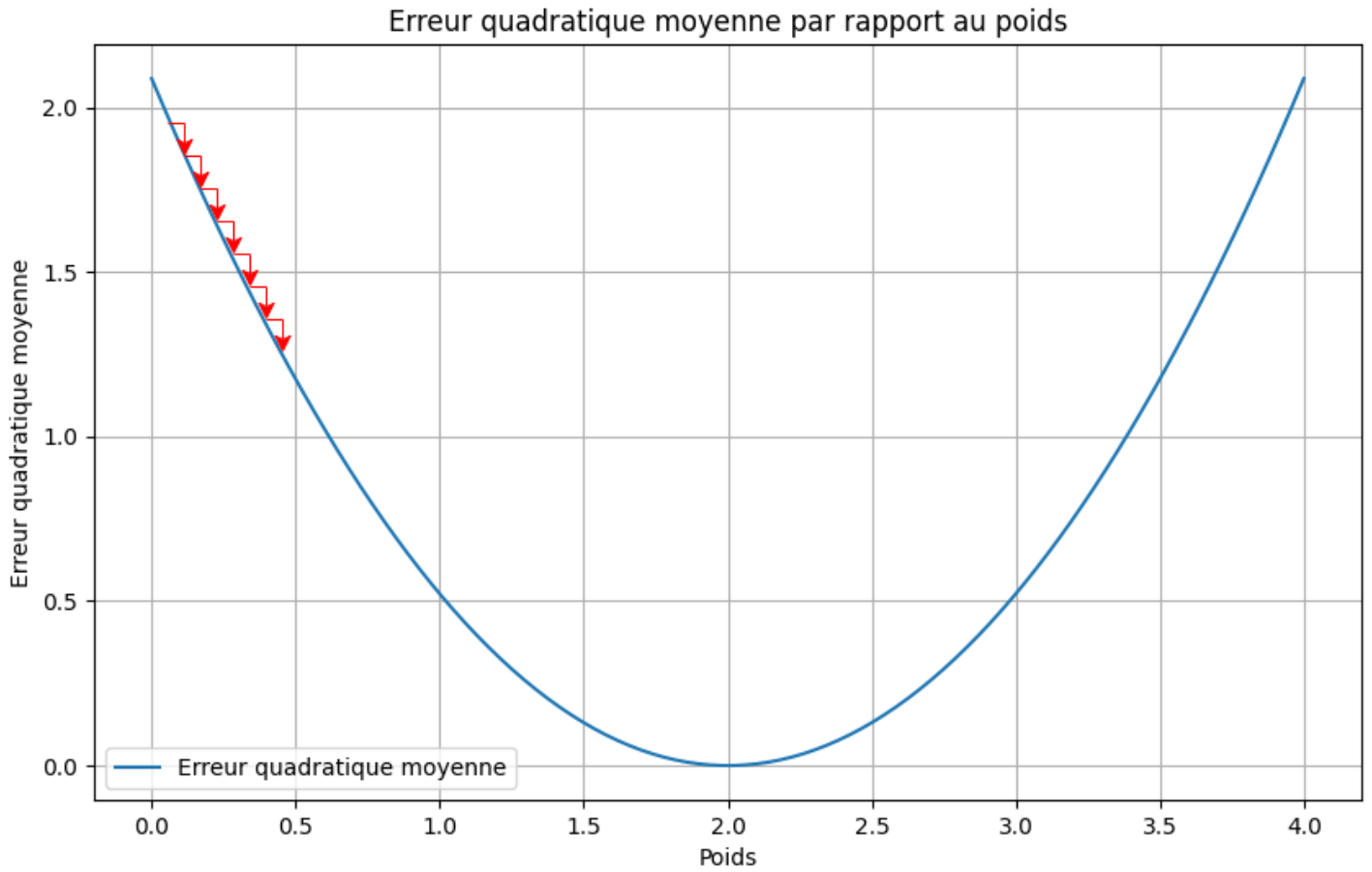
et

$$\frac{\partial \mathcal{C}}{\partial b} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)$$

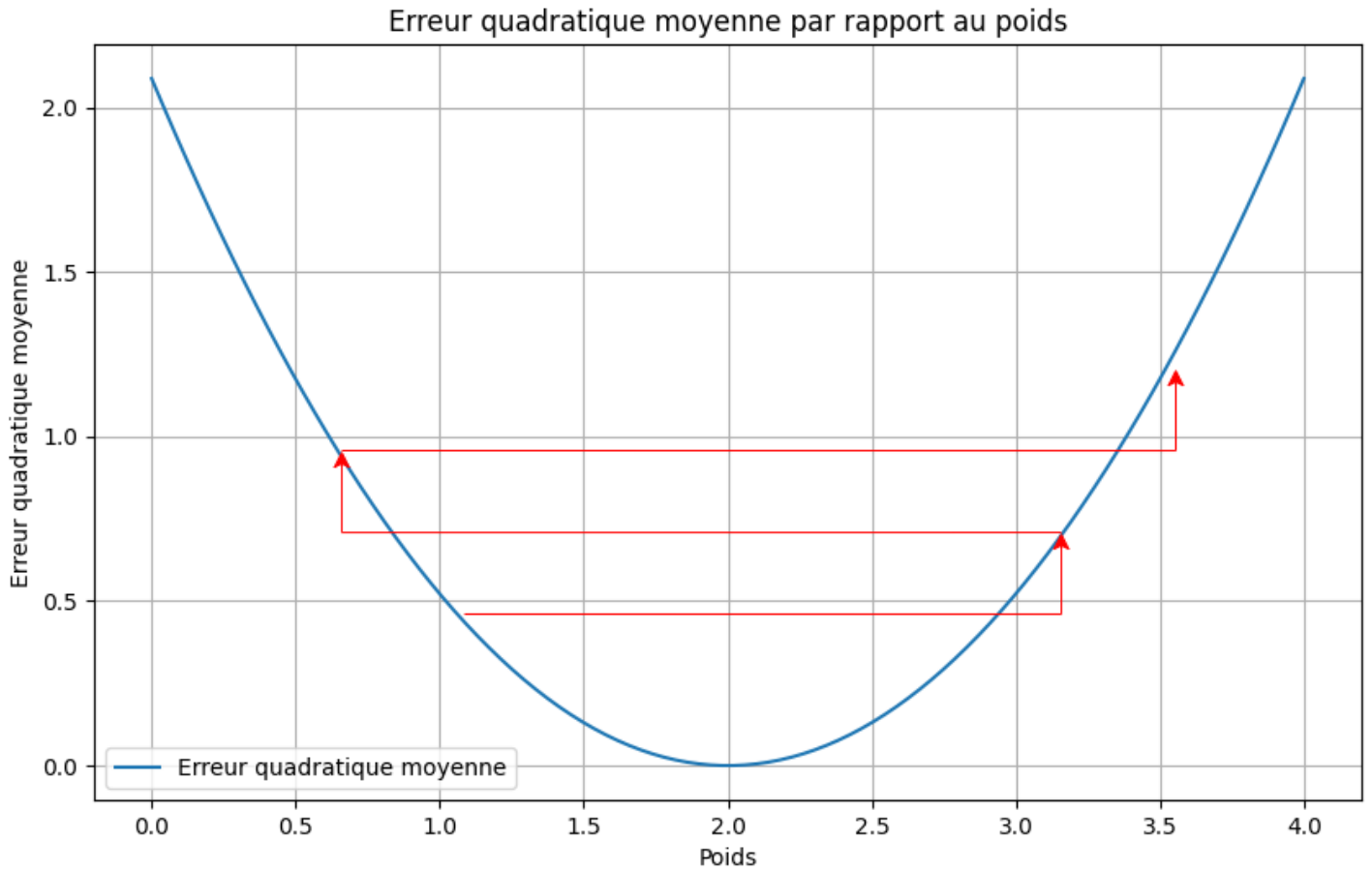
Taux d'apprentissage

- Taux d'apprentissage : unité de correction de l'erreur
- `w -= gradient(w) * lr`
- Importance du choix du taux d'apprentissage (learning rate)
- Exemple avec taux d'apprentissage faible et élevé

Trop faible


Erreur quadratique moyenne par rapport au poids
Erreur quadratique moyenne
Poids
Erreur quadratique moyenne
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

Trop élevé


Erreur quadratique moyenne par rapport au poids
Erreur quadratique moyenne
Poids
Erreur quadratique moyenne
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

Epoch et Batches

Batches

- Sous-ensemble des données pour la mise à jour des paramètres
- Avantages: Réduction de la mémoire, accélération du calcul, convergence rapide

Epoch

- Un passage complet de l'ensemble des données d'entraînement
- Répétition pour améliorer les performances

Implémentation en Python

Plus loin: Réseau de Neurones Feed-Forward

MNIST Dataset

- 60,000 images de chiffres de 0 à 9
- Images en nuances de gris de 28x28 pixels
- Chaque image: 784 valeurs (pixels)
- Conversion des valeurs de pixels de [0, 255] à [0, 1]

Fonction d'Activation: Softmax

- Transformation des scores en probabilités

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

où x_i est le score de la classe i et le dénominateur est la somme des exponentielles de tous les scores.

Fonction de Perte: Entropie Croisée

- Adaptée à la classification multi-classes

$$L = -\sum_{i=1}^{C} y_i \log(p_i)$$

où :

- C est le nombre de classes.
- y_i est 1 si la classe réelle est i et 0 autrement (one-hot encoding).
- p_i est la probabilité prédite pour la classe i.

Rétropropagation dans le Réseau Feed-Forward

Structure du Réseau

- Couche d'entrée: 784 entrées
- Couche de sortie: 10 neurones
- Poids: matrice 10x784
- Biais: vecteur 10

Forward et Backwards

1. Calcul des logits et application de Softmax
2. Calcul des gradients et mise à jour des paramètres

Gradient par rapport aux poids (w):

$$\frac{\partial L}{\partial w} = \frac{1}{n} \cdot X^T \cdot (\hat{Y} - Y)$$

Gradient par rapport aux biais (b):

$$\frac{\partial L}{\partial b} = \frac{1}{n} \cdot \sum (\hat{Y} - Y)$$

Implémentation en Python