Graphical Models

Restricted Boltzmann Machine

Nawal Bendjelloul

Université Paris-Saclay
Christophe Ambroise

9 février 2024

Introduction

Restricted Boltzmann Machine - RBM

- **réseau de neurones probabiliste** pour l'apprentissage non supervisé
- **extrait** des **caractéristiques pertinentes** à partir de données complexes.
- exemple d'applications : génération de données, classification

Objectifs

- **Comprendre** le concept des RBM
- **Implémenter** une RBM de zéro
- **Analyser** et **comparer** les résultats.

Introduction

Les RBM

- réseaux de neurones **stochastiques**
- constituées de **deux couches** : une visible et une cachée, connectées par poids
- apprentissage par **contrastive divergence**

$\rightarrow$ *but principal* : apprendre une **représentation utile des données** en modélisant les relations entre les variables d'entrée

Structure d'une RBM

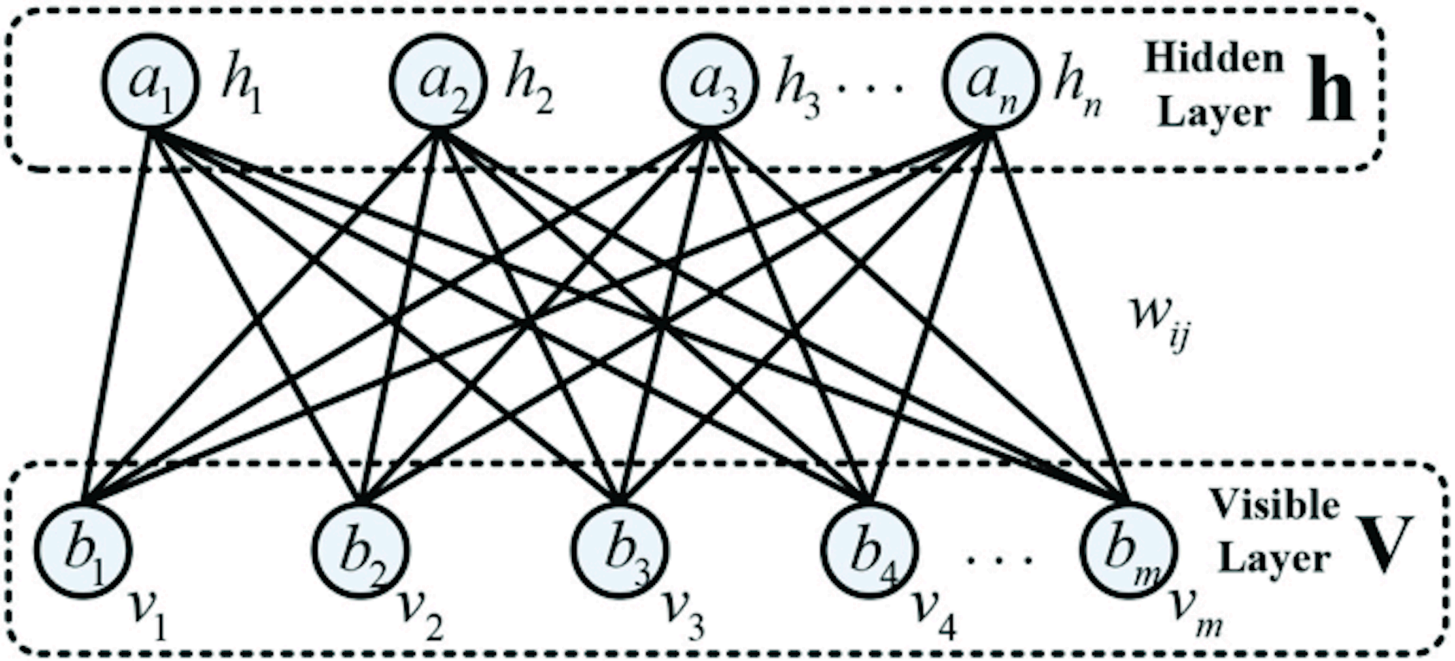

Figure 1 – Structure d'une RBM

Fonctionnement des RBM

Phase positive

- Activation neurones cachés selon **image d'entrée**
- **Reconstruction image** à partir neurones cachés

Phase négative

- Activation neurones cachés selon **image reconstruite**
- **Reconstruction nouvelle image** à partir neurones cachés.

Contrastive divergence

- Calcul **erreur de reconstruction**
- **Mise à jour** des poids

$\rightarrow$ **fonction d'énergie** à minimiser pour optimiser configuration du réseau

Fonction d'énergie

→ **fonction d'énergie d'une RBM** :

$$E(v,h) = -\sum_i a_i v_i - \sum_j b_j h_j - \sum_{i,j} v_i w_{ij} h_j$$

où,

- a_i, b_j : biais des neurones visibles et cachés.
- v_i, h_j : états des neurones visibles et cachés.
- w_{ij} : poids entre neurones visibles et cachés.

→ **probabilité d'une configuration** :

$$P(v,h) = \frac{e^{-E(v,h)}}{Z}$$

Objectif

MNIST

- 70 000 images de **chiffres manuscrits**
- taille 28x28 pixels
- ***preprocessing*** : normalisées + vecteurs binaires

→ *objectif* : **reconstruire** les images MNIST via une RBM

Etapes de la classe `RBM`

1. **Initialisation** poids et biais
2. Calcul probabilités d'activation pour neurones cachés - ***phase positive***
3. Reconstruction données visibles à partir neurones cachés - ***phase négative***
4. **Mise à jour** poids et biais pour minimiser erreur de reconstruction

Initialisation de la Classe RBM

Initalisation

- poids : avec distribution **normale**
- biais visibles et cachés : à **zéro**

Les choix

- learning rate : 0.05
- batch size : 87
- epochs : 20
- nombre de neurones visibles et cachées : 784 | 64
- fonction d'activation `sigmoid`

Phase positive

```
1 def _sample_hidden(self, visible):
2     activations = np.dot(visible, self.weights) + self.hidden_bias
3     probabilities = self._sigmoid(activations)
4     return np.random.rand(*probabilities.shape) < probabilities
```

- calcule les activations des neurones cachés à partir des entrées
- convertit les activations en probabilités d'activation
- échantillonne les états cachés selon probabilités

$\rightarrow$ détermine comment les entrées sont représentées par les neurones cachés

Phase négative

```
1 def _sample_visible(self, hidden):
2     activations = np.dot(hidden, self.weights.T) + self.visible_bias
3     probabilities = self._sigmoid(activations)
4     return np.random.rand(*probabilities.shape) < probabilities
```

- utilise les états cachés pour reconstruire les entrées
- transforme les activations en probabilités de réactivation des neurones visibles
- échantillonne les états visibles reconstruits selon probabilités

→ important pour affiner les poids et améliorer la reconstruction

Mise à jour des poids

```
1 def _update_weights(self, batch):
2     # phase positive : entrées -> états cachés
3     _, hidden_samples = self._sample_hidden(batch)
4     # phase négative : états cachés -> reconstructions
5     visible_recon_probabilities, _ = self._sample_visible(hidden_samples)
6     positive_phase = np.dot(batch.T, hidden_samples)
7     negative_phase = np.dot(visible_recon_probabilities.T, hidden_samples)
8     # mise à jour des poids et biais
```

- calcule la différence entre les phases positive et négative pour ajuster les poids
- mis à jour des biais pour réduire l'erreur de reconstruction

→ améliore progressivement génération des reconstructions

Erreur de reconstruction

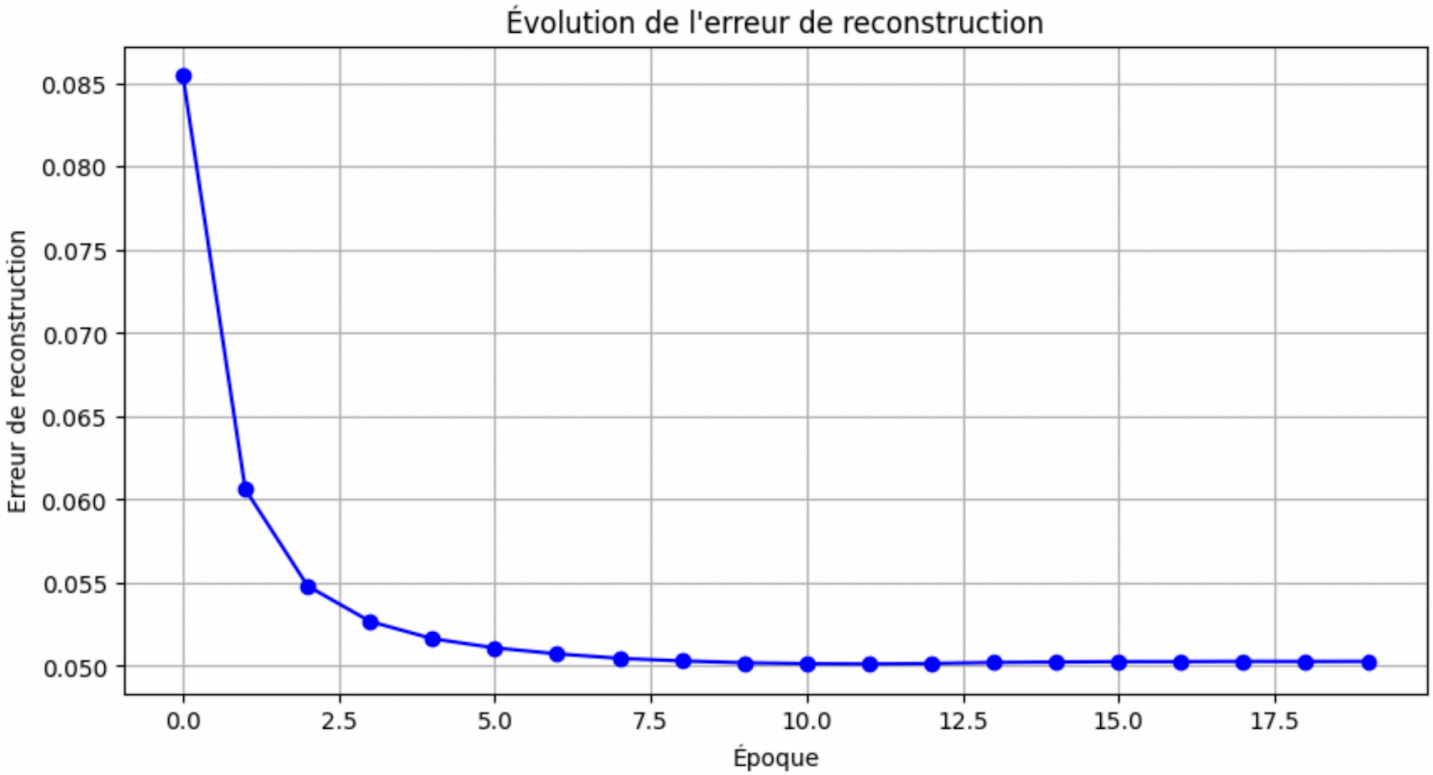

Figure 2 – Évolution de l'erreur de reconstruction

Train : 0.050 **Test** : 0.049

sklearn Train : 0.071 **sklearn Test** : 0.071

Comparaison - reconstruction des images de test

Figure 3 – exemple d'images reconstruites avec notre RBM

Figure 4 – exemple d'images reconstruites avec sklearn

Comparaison - Composants extraits par RBM

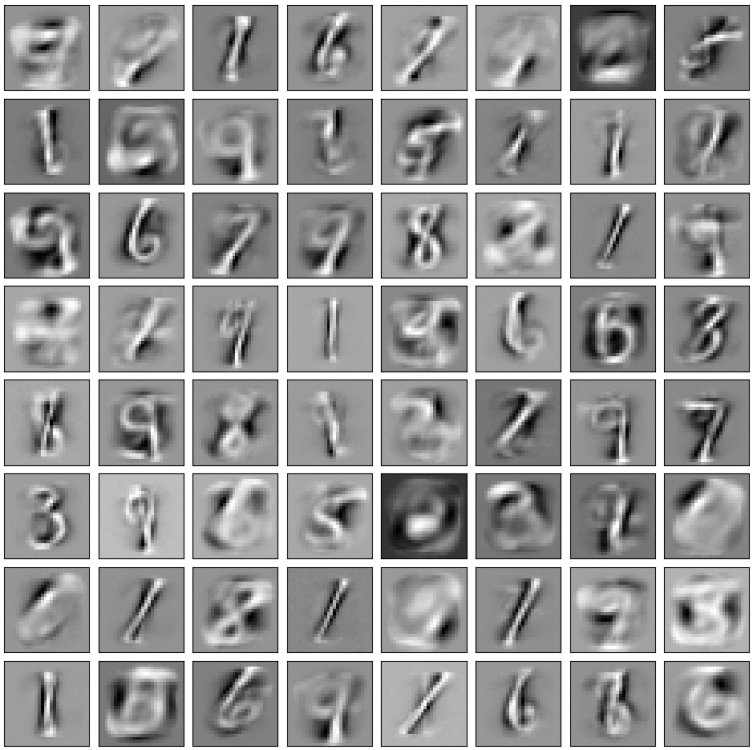

Figure 1 : notre RBM

Figure 2 : avec sklearn

Conclusion

- **Performance**
  $\rightarrow$ faible erreur de reconstruction

- **Composants extraits**
  $\rightarrow$ caractéristiques des chiffes identifiés

- **Limites**
  $\rightarrow$ bruit : optimisation hyperparamètres nécessaire

Références I

[1] *A Practical Guide to Training Restricted Boltzmann Machines.*
Geoffrey Hinton (2010)

[2] *Restricted Boltzmann Machine (RBM) with Practical Implementation.*
Amir Ali

[3] *Restricted Boltzman Machine from Scratch.*
Alin Cijov

[4] *Small binary RBM on MNIST.*
Jan Melchior