Homework 3: Unsupervised Learning

Part 1: Classification with K-means algorithm

The K-means algorithm is a fundamental tool among the unsupervised learning model. Consider a problem with a dataset $\mathcal{D} = \{x_i\}_{i=1}^n$ where $x_i \in \mathbb{R}^d$ with no labels, we are aiming at finding some hidden structure within the data, namely, we would like to find clusters in the dataset. Classifiers have been studied in TP but mainly for supervised learning, here the data are not labeled.

The K-means algorithm tries to classify the dataset in K clusters. Each cluster is represented by a centroid, meaning the average of the points within the cluster. We note C_k the set of points of a cluster k and μ_k its centroid. Then, the algorithm minimizes the intra-cluster variance, in other words, it tries to reduce the distance between the points of the cluster and the centroid.

More technically, the algorithm works iteratively in two main steps:

- Points are assigned to clusters based on their proximity to existing centroid
- Centroids are updated by taking the average of the points assigned to each cluster.

The Loss function of the problem can be written as:

$$J(\mu_1, \dots, \mu_K) = \sum_{i=1}^{n} \|x_i - \mu(i)\|^2 \ ,$$

where $\mu(i)$ is the centroid of the cluster assignated to x_i.

We want to check the understanding of the choice of this loss function. Does it match the aforementioned rules? Then, could it help to understand if the algorithm converges?

Additionnally, until Question 8, we assume that the clusters C_k are disjoint, especially at initialization.

The Secret is in the Notation $\mu(i)$

The confusion usually comes from the term $\mu(i)$.

- μ_k (with a subscript k) refers to a specific centroid (e.g., Centroid #1, Centroid #2).
- $\mu(i)$ (function of i) is a shorthand way of saying: **"The centroid that point x_i currently belongs to."**

A Concrete Example

Imagine you have **3 Data Points** (x_1, x_2, x_3) and **2 Clusters** (Centroids μ_1, μ_2).

- x_1 belongs to Cluster 1.
- x_2 belongs to Cluster 1.
- x_3 belongs to Cluster 2.

The formula $J = \sum_{i=1}^{3} \|x_i - \mu(i)\|^2$ expands like this:

1. **For $i = 1$:** x_1 is in Cluster 1, so $\mu(1)$ is μ_1. Term: $\|x_1 - \mu_1\|^2$
2. **For $i = 2$:** x_2 is in Cluster 1, so $\mu(2)$ is μ_1. Term: $\|x_2 - \mu_1\|^2$
3. **For $i = 3$:** x_3 is in Cluster 2, so $\mu(3)$ is μ_2. Term: $\|x_3 - \mu_2\|^2$

Total J:

$$J = \|x_1 - \mu_1\|^2 + \|x_2 - \mu_1\|^2 + \|x_3 - \mu_2\|^2$$

Summary

- The sum is over i because we calculate the error **point-by-point**.
- $\mu(i)$ dynamically grabs the correct μ_k for that specific point.

Next Step: This concept is exactly what **Question 1** asks you to prove. Would you like to see how to transform this "point-wise" sum ($\sum_i$) into the "cluster-wise" sum ($\sum_k$)?

Question 1:

First, let us familiarize ourselves with the loss function:

- Prove the second equality:

$$J(\mu_1, \dots, \mu_K) = \sum_{i=1}^{n} \|x_i - \mu(i)\|^2 = \sum_{k=1}^{K} \sum_{x_i \in C_k} \|x_i - \mu_k\|^2$$

- What does the term $\sum_{x_i \in C_k} \|x_i - \mu_k\|^2$ represent?

- Explain why this form of the loss function is more convenient.

Q1 Ans

Question 2:

Let us focus on the first point of the algorihtm, let us consider a single point x_i and add the time dependency. Also, we denote by a $'$ the variables after the new assignement of the data points.

So that, the variables are denoted by: $(\cdot)^t \to (\cdot)^{t\,\prime} \to (\cdot)^{t+1} \to (\cdot)^{t+1\,\prime} \to (\cdot)^{t+2}$

Thus, at each step time t, the new assignements of the variables leads to: (no proof required)

$$\mu^t(i)' = \mathrm{argmin}_{\mu \in \{\mu_k^t\}_k} \|x_i - \mu\|^2$$

For instance, at time t before new assignements, the vector x_i belongs to a cluster k, while after the next assignement, it now belongs to the cluster k' (it can be the same or different from the cluster k).

Thus, for the whole dataset, the algorithm updates the assignement as: $\{\mu^t(i)\} \to \{\mu^t(i)'\}$.

Compare $\|x_i - \mu^t(i)\|^2$ and $\|x_i - \mu^t(i)'\|^2$ for a given point.

Question 3:

We recall that we denote by a $'$ the variables after the new assignement of the data points, so that: $\mu^t(i) \to \mu^t(i)'$ and $J_t \to J_t'$

Thanks to the previous question, compare J_t and J_t'.

Hint: Pick the right formula between the two given for J

Question 4:

Then, we can go ahead by studying the second point of the algorithm. It wants to minimize the intra-cluster variance:

$$\{\mu_k^{t+1}\}_k = \mathrm{argmin}_{\{\mu_k\}_k \in \mathbb{R}^d} J_t'\left(\{\mu_k\}_k\right) = \mathrm{argmin}_{\{\mu_k\}_k \in \mathbb{R}^d} \sum_{k=1}^{K} \sum_{x_i \in C_k^{t\prime}} \|x_i - \mu_k\|^2$$

Show that the optimization can be done cluster-wise:

$$\mu_k^{t+1} = \mathrm{argmin}_{\mu_k \in \mathbb{R}^d} \sum_{x_i \in C_k^{t\prime}} \|x_i - \mu_k\|^2$$

Question 5

Show that the new centroids of time $t+1$ are computed according to the following equality:

$$\mu_k^{t+1} = \frac{1}{|C_k^{t\prime}|} \sum_{x_i \in C_k^{t\prime}} x_i$$

Does it correspond to what you expected from the algorithm?

Question 6:

If we focus on a cluster k at time t after the assignement, noted $C_k^{t\prime}$, could you compare $\sum_{x_i \in C_k^{t\prime}} \|x_i - \mu_k^t\|^2$ and $\sum_{x_i \in C_k^{t\prime}} \|x_i - \mu_k^{t+1}\|^2$?

What can you say about J_t' and J_{t+1} ? Hint: Use the right formula between the two given for J.

Question 7:

Putting together the Questions 3 and 5, compare J_t and J_{t+1}

Question 8:

After recalling a trivial lower bound for the sequence $(J_t)_{t \geq 0}$, what can you say about the convergence?

Question 9:

We just proved that the algorithm converges, but what about its stability:

Let us suppose that the data are sampled from a mixture of K Gaussian, where the choice of K is free for this question. Do you imagine a situation where the algorithm does not classify the data at all? Please design and explain the situation as clearly as possible.

Question 10:

What can you say about those configurations of centroids? What does it imply concerning the minima? Conclude your arguments by discussing the convexity of the problem.

Question 11:

We can also quickly generalize our algorithm.

In some situation, you are aiming at favoring some directions in your data and penalizing the others, so that you can weigh the euclidean distance according to:

$$d^{(w)}(x_i, \mu(i)) = \frac{\sum_{j=1}^{d} w_i (x_{ij} - \mu(i)_j)^2}{\sum_{j=1}^{d} w_j}$$

Show that with a change of variables, the problem remains the same.

Part 2: Restricted Boltzmann Machine

Introduction

The Boltzmann Machine have been inspired by thermodynamic and statistical physics models, more precisely they are part of the Energy Models using the well known Boltzmann Distribution as written in physics style:

$$P(E) = \frac{1}{Z} \exp\left(-\frac{E}{k_b T}\right)$$

It becomes in statistical inference framework:

$$P(\mathbf{v}|J, \mathbf{b}) \propto e^{\mathbf{v}^T J \mathbf{v} + \mathbf{b}^T \mathbf{v}} = e^{-E(\mathbf{v})}$$

where:

- $\mathbf{v} \in \mathbb{R}^n$: The binary vector with components $v_i = 0 \text{ or } 1$

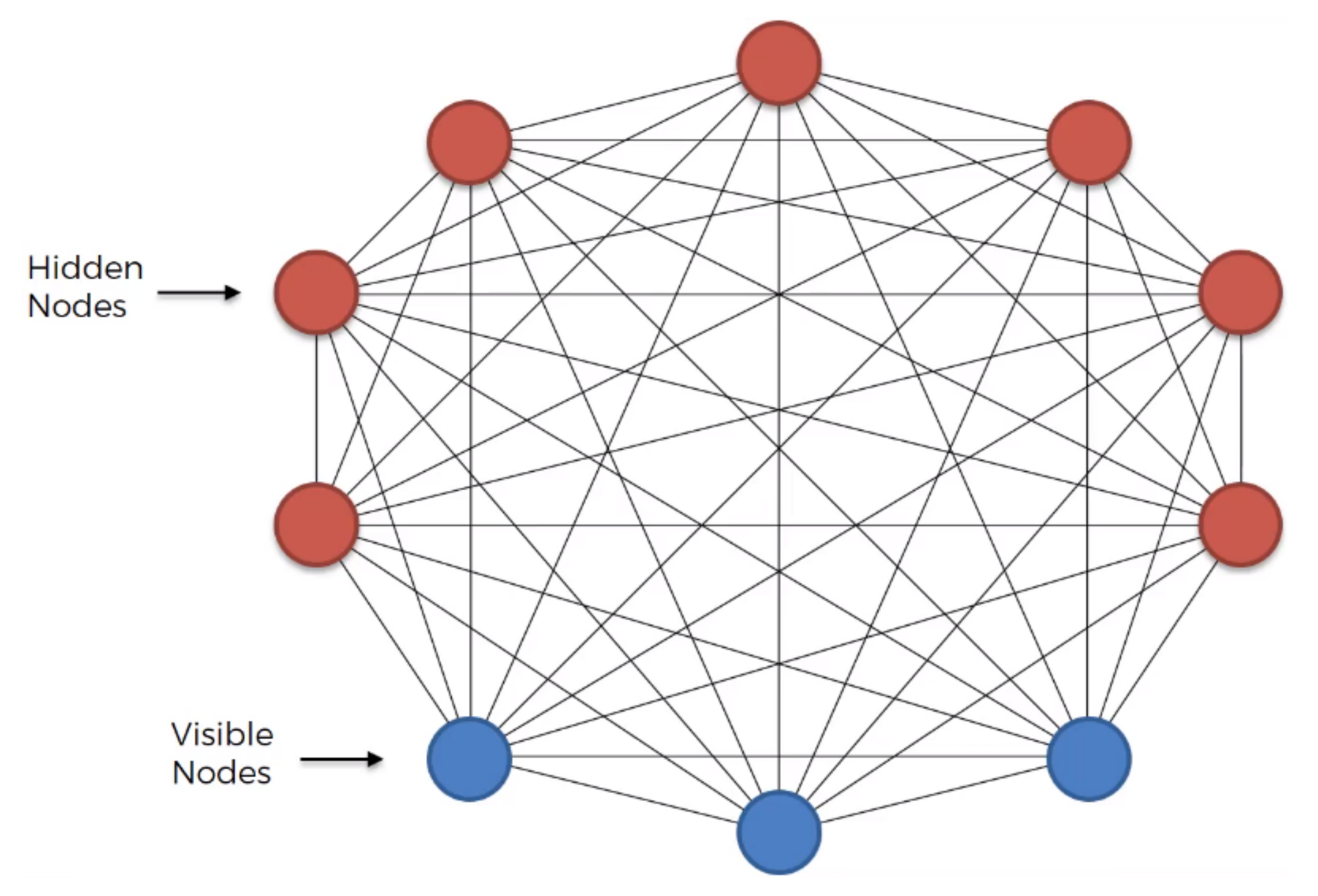
- $J \in \mathbb{R}^{n \times n}$: The coupling matrix

- $\mathbf{b} \in \mathbb{R}^n$: Field

- $E(\mathbf{v}) \in \mathbb{R}$: Energy

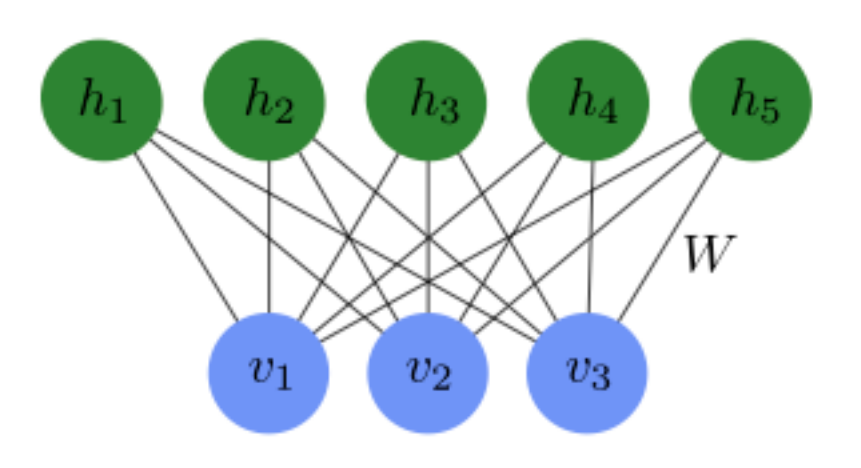
However, one problem arised with initial Boltzmann Machine (BM) -- like its parent models in statistical physics (as the SK model) -- all the units are interacting through complicated dependencies. For example, if we consider 3 components of $\mathbf{v}$: v_1, v_2, and v_3, there are trivial interactions such as one modelised by $P(v_1, v_2)$ corresponding to the correlation between the two first components of $\mathbf{v}$, but there are also none trivial interactions. Indeed, if some term like $P(v_1, v_2|v_3)$ which suggests that the correlation x_1 and v_2 depends on v_3 and this is clearly none linear.

A really ingenious way to overcome this situation is to replace all the tricky interactions between the units $\mathbf{v} \in \mathbb{R}^n$ by connections through hidden units $\mathbf{h} \in \mathbb{R}^m$, artifically created. Indeed, correlations between two units v_1 and v_2 (specially the dependency of their correlations on other units v_3, v_4,...) can be atrificially replaced by introducing a third unit h_1 and considerin only linear correlations between $v_1 \leftrightarrow h_1$, $h_1 \leftrightarrow v_2$ and $v_1 \leftrightarrow v_2$. The units v_i are now called the visible units. This model is the most known version of BMs.



However, this model is still fully connected and makes the computation really costful. Then, one can even simplify the model by considering zero intra layer interractions. This simplified model is call Restricted Boltzmann Machine (RBM) (Physics Nobel Price 2024 🥳).

Thus, the RBM architecture consists of two layers of binary stochastic units: a $\textbf{visible layer } \mathbf{v}$ and a $\textbf{hidden layer } \mathbf{h}$. The layers are fully connected, but there are no connections within a layer, making the model a $\textbf{bipartite graph}$.



Restricted Boltzmann Machines (RBMs) are a class of energy-based probabilistic graphical models that are commonly used in machine learning for tasks such as dimensionality reduction, feature learning, and generative modeling.

Energy Function and Probabilities

The joint configuration of the visible units $\mathbf{v} \in \{0,1\}^d$ and the hidden units $\mathbf{h} \in \{0,1\}^m$ is associated with an $\textbf{energy function}$, defined as:

$$E(\mathbf{v}, \mathbf{h}) = -\mathbf{v}^\top \mathbf{W} \mathbf{h} - \mathbf{b}^\top \mathbf{v} - \mathbf{c}^\top \mathbf{h}$$

where:

- $\mathbf{W} \in \mathbb{R}^{d \times m}$ is the weight matrix connecting the visible and hidden units,
- $\mathbf{b} \in \mathbb{R}^d$ field of the visible units or also called the biases of the visible units,
- $\mathbf{c} \in \mathbb{R}^m$ field of the hidden units of also called the biases of the hidden units.

The energy function determines the joint probability distribution over $\mathbf{v}$ and $\mathbf{h}$:

$$P(\mathbf{v}, \mathbf{h}) = \frac{1}{Z} \exp(-E(\mathbf{v}, \mathbf{h}))$$

where Z is the $\textbf{partition function}$, ensuring normalization:

$$Z = \sum_{\mathbf{v}, \mathbf{h}} \exp(-E(\mathbf{v}, \mathbf{h}))$$

The marginal probability of the visible units $\mathbf{v}$ is obtained by summing over all possible configurations of the hidden units:

$$P(\mathbf{v}) = \frac{1}{Z} \sum_{\mathbf{h}} \exp(-E(\mathbf{v}, \mathbf{h})).$$

Question 12:

Write a valid expression of the energy $E(\mathbf{v}, \mathbf{h})$ in the case of a BM (non-restricted) with an hidden layer.

Conditional Independence

Question 13:

One of the key properties of RBMs is the $\textbf{conditional independence}$ between units within a layer:

Compute the conditional probability and show that:

$$P(h_j = 1|\mathbf{v}) = \sigma\left(c_j + \sum_i v_i W_{ij}\right)$$

and

$$P(v_i = 1|\mathbf{h}) = \sigma\left(b_i + \sum_j h_j W_{ij}\right)$$

where $\sigma(x) = \frac{1}{1+e^{-x}}$ is the sigmoid activation function.

This bipartite structure enables efficient Gibbs sampling for approximating the intractable joint distribution $P(\mathbf{v}, \mathbf{h})$.

Learning in RBMs

Question 14:

Training an RBM involves maximizing the likelihood of the data distribution. To do so we are aiming at using a gradient descent/ascent on the weights (and biases).

Compute the log-likelihood $\mathcal{L}(\mathbf{v})$, remember that the model is part of the unsupervised learning.

Question 15:

Compute the gradient of the log-likelihood with respect to the weights $\mathbf{W}$ and the biases $\mathbf{b}$, $\mathbf{c}$:

Now, it should be possible to implement the RBM!

Question 16: (Open question)

While it seems possible to run RBM algorithm, note that the second term in the gradient w.r.t. $\mathbf{W}$ is computationally expensive due to the intractability of Z, the approximation Contrastive Divergence - k is often use. Research what is this approximation, is this approximation enough, why? Explain it with your own words and cite the papers you used for your documentation.

Applications of RBMs

RBMs are widely used in tasks such as:

- $\textbf{Dimensionality reduction}$: Similar to PCA but capable of capturing non-linear structures,
- $\textbf{Feature learning}$: For pre-training deep neural networks,
- $\textbf{Collaborative filtering}$: Used in recommendation systems.