DEEP BELIEF NEURAL NETWORKS (DBN) WITH IMPLEMENTATION IN PYTHON AND MATLAB

What is DBN ?

Deep Belief Network is stochastic deep learning model that works benefitting the power of **RBMs (Restricted Boltzman Machines)** in order to provide efficient solutions for supervised and unsupervised tasks.

To better understand the architecture and working procedure of DBN, let us investigate the **RBMs and Boltzman Machines (BMs)** step by step in detail.

Restricted Boltzman Machine (RBM)

Restricted Boltzman Machine (RBM) is a 2 layer (1 visible – 1 hidden layer) generative stochastic energy-based model designed to capture patterns and features in data. Let us have a look at its general architecture and working principle:

1. **Structure:**
    a. The RBM consists of two layers of units: the visible layer and the hidden layer.
    b. The visible layer represents the input data, like pixels in an image or words in text.
    c. The hidden layer captures abstract features that help explain the input data.

2. **Energy Function:**
    a. RBMs use an energy function to calculate how well the visible and hidden units match each other.
    b. Lower energy means the units are in harmony, representing a likely pattern or feature.

3. **Pretraining:**
    a. RBMs are pretrained in an unsupervised manner to learn these features from data.
    b. Pretraining involves adjusting weights and biases to make RBM like the data's patterns.

4. **Contrastive Divergence (CD):**
    a. Contrastive Divergence is a method to update the weights and biases during training.
    b. It compares the data's energy with the energy of generated samples, adjusting the model.

5. **Feature Learning:**
    a. The RBM learns to recognize important patterns and features from the data.
    b. These learned features are captured in the hidden layer's activations.

6. **Applications:**
    a. RBMs are used for various tasks like image recognition, recommendation systems, and feature learning.
    b. They serve as building blocks for more complex models like Deep Belief Networks.

Let us investigate the each section one-by-one.

1. **Structure:**
    - An RBM must consist of **2 different** layers: **1 visible layer** and **1 hidden layer.**
    - Visible and hidden units are connected to each other in **bi-directional way (data can flow from visible-to-hidden and hidden-to-visible units)**.
    - There is **no connection** between the units **within the "same layer".**
    - Number of nodes within the layers is **not restricted. However,** number of nodes within the **input layer shoud match with the dimensionality** of our data. **E.g:** gray-scale data with pixels of **28x28 requires 784 input nodes**.

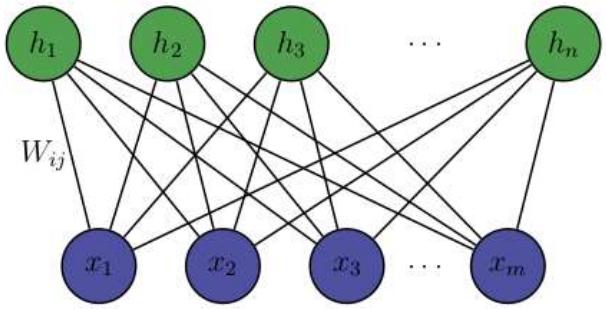


Figure 1: Graphical description of Restricted Boltzmann Machine

- **Visible Layer:**
    - The part that directly **interacts with the outside world**. For example, if you're using an RBM to analyze images, the visible layer might correspond to the **pixels of the image**. If you're working with text data, it could be the **words in a sentence**.
- **Hidden Layer:**
    - The hidden layer is a bit more abstract. It doesn't directly see the outside world, but instead, it **tries to learn patterns and features from the visible layer**.. For example, in an image analysis task, the hidden layer might learn to recognize edges, shapes, or even higher-level patterns like faces.

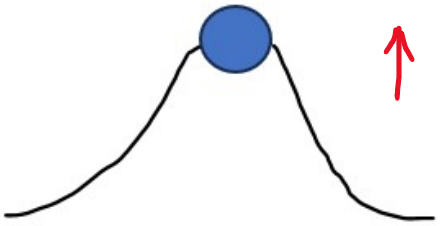
2. **Energy Function**

By **energy-based model,** we mean sort of machine learning approach that quantifies the **compatibility of different configurations** of variables using an energy function. **Just as our brain's way of recognizing complex things, it recognize the features and patterns of data.**

In the context of RBMs, this energy function is used to calculate the **harmony** or **compatibility** between the visible and hidden units' states.

To better understand the concept of energy, let us imagine the status of the ball on a hill.

- The height of the hill represents the "energy" of the ball at that position,
- The ball wants to settle in a position where its energy is lowest.



(figure 1)

(figure 2)

High Energy: If the ball is at a high point on the hill, it has high potential energy. It wants to roll down to a lower point where its energy is minimized and its position is stable. (figure 1)

Low Energy: When the ball is at the bottom of the hill, it has low potential energy. It's in a stable state, and it doesn't want to move unless something external happens. (figure 2)

Now, let's relate this analogy to RBMs:

Visible Units (Data): Think of these as the ball's position on the hill. Different positions represent different data configurations (e.g., pixel values in an image).

Hidden Units (Features): Imagine these as the hill's landscape. Hidden units are like certain characteristics or features that help describe the visible data. Each hidden unit might represent a different shape or pattern.

Mathematical Expression of Energy and Likelihood Function

In RBMs, the energy function calculates a value based on the compatibility of the visible and hidden units. It's like determining the height of the hill based on the position of the ball. Lower energy values mean the configuration of visible and hidden units is more likely.

The energy function and the likelihood function of the RBM are expressed as:

$$E(\mathbf{v}, \mathbf{h}; \theta) = -\sum_{i=1}^{D} \sum_{j=1}^{J} v_i W_{ij} h_j - \sum_{j=1}^{J} b_j h_j - \sum_{i=1}^{D} c_i v_i, \tag{1}$$

$$\begin{aligned} P(\mathbf{v}; \theta) &= \sum_{\mathbf{h}} P(\mathbf{v}, \mathbf{h}; \theta) = \sum_{\mathbf{h}} \left(\frac{1}{Z(\theta)} \exp\left(-E(\mathbf{v}, \mathbf{h}; \theta)\right) \right) \\ &= \frac{1}{Z(\theta)} \sum_{\mathbf{h}} \exp(-E(\mathbf{v}, \mathbf{h}; \theta)), \end{aligned} \tag{2}$$

where $v_i \in \{0, 1\}$, $h_j \in \{0, 1\}$, $\mathbf{W} = (W_{ij}) \in R^{D \times J}$ is the weights connecting visible units and hidden units, $\mathbf{c} = \{c_i\}_{i=1}^{D}$ is the bias terms of the visible layer, $\mathbf{b} = \{b_i\}_{i=1}^{J}$ is the bias terms of the hidden layer, $Z(\theta) = \sum_{\mathbf{v}} \sum_{\mathbf{h}} \exp(-E(\mathbf{v}, \mathbf{h}; \theta))$ is the partition function, and $P(\mathbf{v}; \theta)$ also can be called the marginal distributions of $P(\mathbf{v}, \mathbf{h}; \theta)$.

Energy and likelihood value of each node is calculated specifically according to the given formula and each node (unit) is decided to be in the state of “on” (1) or “off” (0). The important thing to keep in mind here is that, for every single node in the RBM, we calculate the energy. Then, by using this result within the formula of probabilistic value calculation, we decide if the staets of nodes will be set to “on” (1) or “off” (1).

If the result of probability calculation function is greater than a “random value” (where random value is a different value for all the nodes within the network, this is one of the features that makes this model stochastic), then it will be set as on (1) and off (0) for otherwise.

3. Gibbs Sampling

Gibbs Sampling is a **Markov Chain Monte Carlo (MCMC)** technique used to generate new configurations **(states)** of visible and hidden units in an RBM based on the calculated probabilities.

To better understand this definition, lets consider our current status in training process. Right now, we have our nodes in a certain state (some are “on” and some are “off”). Using the Gibbs sampling, we will set these nodes to a diverse combination of states (again based on a random number). Meaning that, this time some different nodes will be on and some different nodes will be off after this sampling.

With such approach, each node will have a chance to be in the state of both “on” and “off” and model will be able to learn patterns and features for different configurations of states.

This “random” status settings will be performed for both visible and hidden layer.

1- **Positive Phase**:
   - Start with the actual input data as the initial state for the visible layer.
   - Calculate the activation probabilities for each hidden unit based on the energy function and the states of the visible units.
   - Make stochastic decisions for each hidden unit's state based on these activation probabilities.
   - The hidden unit states in the positive phase are determined by the visible unit states.

2- **Negative Phase:**
   - Use the states of the hidden units obtained from the positive phase as the initial state for the hidden layer.
   - Calculate the activation probabilities for each visible unit based on the energy function and the states of the hidden units.
   - Make stochastic decisions for each visible unit's state based on these activation probabilities.
   - The visible unit states in the negative phase are determined by the hidden unit states.

So, in both phases, the states of one layer are used to probabilistically determine the states of the other layer. (number of iterations for the sampling can be defined during hyper-parameter tuning if a comprehensive library is provided for model creation).

3- **Contrastive Divergence (CD-k)**

Contrastive Divergence refers to the process of calculating the **difference between the expected values of certain products of activations in the positive and negative phases**. This difference is used to **update the model's weights and biases** in a way that it will **reduce the divergence between the two (positive and negative) phases** in order to align the model's distribution with the data distribution, improving its ability to learn and represent patterns in the data.

“k” value here corresponds the number of times Gibbs sampling is performed. Even though technically CD-k and Gibbs sampling are 2 seperate processes, because of the historical development of such methods, number of iterations times is indicated within the “CD” method’s name, not that of the “Gibbs Sampling”.

1- **Positive Phase Calculation:**
   - Calculate the probabilities of hidden units being on given the visible units' states:
     - $p(\mathbf{h}|\mathbf{v}) = \sigma(\mathbf{W}\mathbf{v} + \mathbf{b})$
   - Calculate the expected outer product of hidden and visible units:
     - $\langle \mathbf{h}\mathbf{v}^\top \rangle_{\text{data}} = \mathbf{h}_{\text{data}}\mathbf{v}_{\text{data}}^\top$
   - Calculate the expected values of hidden and visible units:
     - $\langle \mathbf{h} \rangle_{\text{data}} = \mathbf{h}_{\text{data}}$
     - $\langle \mathbf{v} \rangle_{\text{data}} = \mathbf{v}_{\text{data}}$

2- **Negative Phase Calculation (Gibbs Sampling):**
   - Start from the visible units' states sampled from the data.
   - Perform k steps of Gibbs sampling
     ('Bernoulli' term is explained in the end)
     - $\mathbf{h} \sim \text{Bernoulli}(\sigma(\mathbf{W}\mathbf{v} + \mathbf{b}))$
     - $\mathbf{v} \sim \text{Bernoulli}(\sigma(\mathbf{W}^\top\mathbf{h} + \mathbf{a}))$
   - Calculate the expected outer product of hidden and visible units:
     - $\langle \mathbf{h}\mathbf{v}^\top \rangle_{\text{model}} = \mathbf{h}_{\text{model}}\mathbf{v}_{\text{model}}^\top$

3- **Weight and Bias Update:**
   - Update the weights and biases using the difference between positive and negative phases:
     - $\Delta\mathbf{W} = \epsilon\left(\langle \mathbf{h}\mathbf{v}^\top \rangle_{\text{data}} - \langle \mathbf{h}\mathbf{v}^\top \rangle_{\text{model}}\right)$
     - $\Delta\mathbf{a} = \epsilon\left(\langle \mathbf{v} \rangle_{\text{data}} - \langle \mathbf{v} \rangle_{\text{model}}\right)$
     - $\Delta\mathbf{b} = \epsilon\left(\langle \mathbf{h} \rangle_{\text{data}} - \langle \mathbf{h} \rangle_{\text{model}}\right)$

Here, ϵ is the learning rate that controls the size of the weight updates (just as it is in traditional ANN's back propagation process).

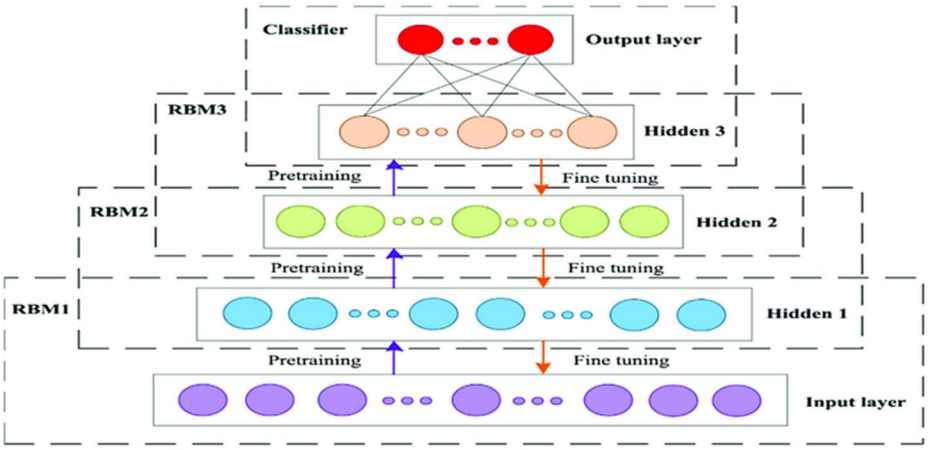
With such approach, features and patterns in the data is learnt in a strong manner. However, this is the working principle of RBM, not the DBN. So, how about the DBN ?

DBN Model Creation

A DBN model can be created by stacking RBMs on top of each other and adding a classifier/regressor to the end of network in order to perform predictions. In a DBN there are a few factors that differ from RBM:

1- While the layers are undirected (bi-directional) in an RBM, layers are one-directed in a DBN except for the top-2 layers of the DBN (which are the hidden layer of the last RBM and regressor/classifier layer of the network.)
2- After the pretraining stage, it requires a fine-tuning with back propagation for a well-optimized regressor/classifier.
3- Gibbs sampling will be performed seperately for each RBM and finalled will be added up inbetween (all negatives and all positives) with the Contrastive divergence (CD- k) method for parameter (bias, weight values) tuning.

A graphical representation of a DBN with 3 stacked-RBMs would look like following:



During the pre-training stage, right after when the energy function and probabilistic value function is calculated for each node just as it is in a single RBM's training stage, Gibbs sampling is performed one-by-one for each RBM in the DBN.

Lets assume we have a DBN model with 3 stacked RBMs. When applying Gibbs sampling, we apply it for the first RBM's visible and hidden layers to find out the positive and negative phase values. Then, we proceed to the second RBM.

As it can be understood from the graphical representation of the DBN, hidden layer of each RBM becomes the visible layer of the next RBM. (E.g. Hidden layer of the first RBM is the visible layer of second RBM. Hidden layer of the second RBM is the visible layer of the third RBM…..and so on) So, we can use the probabilistic values and states of the current RBM's hidden layer as the input for the following RBM's visible layer. Then, same Gibbs sampling process is performed for this RBM as well.

For a DBN model with "N" number of RBMs, this procedure is repeated for N times. Then positive phase value and negative phase value of each RBM's final state is calculated and sum up. Sum up values of positive and negative phases is substracted from each other with CD-k method in order to update the parameters (weights, bias) of network. So far, it is pretty similar to the RBM.

Now, once the pre-training is done, fine-tuning should be performed on the model for regression/classification tasks. We achive this by adding a regressor/classifier after the last layer of the network (aka the hidden layer of last RBM in DBN model).

Lets assume we would like to add a "logistic regressor" after the last layer of RBMs. We simply import the library and perform a 'traditional supervised training' by initializing the weights and biases of the classification layer and by choosing a loss function (e.g. MSE for regression and cross-entropy for classification) to perform back propagation on the entire network in order to update all parameters including those from pretraining.

So, to clarify the sequence of events:

Unsupervised Pretraining: Calculate energy, apply Gibbs sampling, CD-k for each RBM separately, update their parameters.

Fine-Tuning: Add a classification/regression layer on top of the RBM stack. Train the entire DBN using backpropagation with labeled data, updating all parameters, including those from pretraining.

That is how the a DBN model works. One important point to remember is that, so far we have discussed the DBN model with the assumption that it will be used for classification task. (For. E.g, for the likelihood function, we have implemented the sigmoid -binary classificaiton- activation function.) In case of regression, it is necessary to adjust these mathematical functions.

On the other hand, changing the "type" of RBM can help to solve this issue as well. So far, we have used a "Bernoulli-BernoulliRBM" type of RBM to understand the problem where first keyword indicates the type of visible layer in the RBM and second keyword indicates the type of hidden layer in the RBM. For regression tasks, type of RBM that can take continous values and a type of RBM that can calculate the probabilistic values of each node in continuous way can be chosen. For e.g. : Gaussian-GaussianRBM can be a useful choice.

The implementation of a DBN model for classification task with a "BernoulliRBM" of the "scikit-learn" library of Python can be found next to this page (will be added soon). Also a toolbox for Deep Belief Network in MATLAB prepared by a Japan professor Masayuki Tanaka can be used as well for regression tasks after correct adjustments (for type of RBMs and unit architectures for visible-hidden layers) on the code.

Masayuki Tanaka (2023).

DeepNeuralNetwork (https://www.mathworks.com/matlabcentral/fileexchange/42853-deep-neural-network), MATLAB Central File Exchange. Retrieved August 29, 2023.

References:

Y. Chen, X. Zhao and X. Jia, "Spectral–Spatial Classification of Hyperspectral Data Based on Deep Belief Network," in IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, vol. 8, no. 6, pp. 2381-2392, June 2015, doi: 10.1109/JSTARS.2015.2388577.

Nan Zhang, Shifei Ding, Jian Zhang, Yu Xue,
An overview on Restricted Boltzmann Machines,
Neurocomputing,
https://doi.org/10.1016/j.neucom.2017.09.065.
https://www.sciencedirect.com/science/article/pii/S0925231217315849

https://medium.com/machine-learning-researcher/boltzmann-machine-c2ce76d94da5

https://medium.com/swlh/what-are-rbms-deep-belief-networks-and-why-are-they-important-to-deep-learning-491c7de8937a

https://www.simplilearn.com/tutorials/deep-learning-tutorial/what-is-deep-belief-network