
# Deep Learning Assignment
# Assignment 1: Restricted Boltzmann Machine

Mohammad Parsa Dini - Std ID: 400101204

May 2025

## Introduction

This report presents the implementation and analysis of a Restricted Boltzmann Machine (RBM) applied to the MNIST dataset as part of the Deep Learning course assignment at Sharif University of Technology. The objective is to understand the theoretical foundations of RBMs, implement the model, train it on the MNIST dataset, and analyze its performance through loss metrics and generated outputs. The report covers the RBM model, its energy-based formulation, probability distributions, contrastive divergence, the architecture used, and the training process. Additionally, it includes visualizations of the model's output over 20 epochs and generated samples, along with inferences drawn from the code.

## 1 Restricted Boltzmann Machine (RBM)

A Restricted Boltzmann Machine is an undirected, generative probabilistic model with a bipartite graph structure, consisting of visible and hidden layers. The visible layer represents the input data, while the hidden layer captures latent features. The "restricted" aspect refers to the absence of intra-layer connections, simplifying computations.

The energy function of an RBM, for visible units $\mathbf{v}$ and hidden units $\mathbf{h}$, is defined as:

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

where $\mathbf{a}$ and $\mathbf{b}$ are bias vectors for the visible and hidden units, respectively, and $\mathbf{W}$ is the weight matrix connecting the layers.

The joint probability distribution over visible and hidden units is:

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

where $Z$ is the partition function, $Z = \sum_{\mathbf{v},\mathbf{h}} \exp(-E(\mathbf{v}, \mathbf{h}))$, normalizing the distribution.

The marginal probability of the visible units is obtained by summing over all possible hidden unit states:

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

The free energy function, which simplifies computations, is:

$$F(\mathbf{v}) = -\log \sum_{\mathbf{h}} \exp(-E(\mathbf{v}, \mathbf{h})) = -\mathbf{a}^\top \mathbf{v} - \sum_j \log(1 + \exp(\mathbf{W}_j^\top \mathbf{v} + b_j)).$$

## 2 Training RBMs with Contrastive Divergence

Training an RBM involves maximizing the likelihood of the data, which requires computing the gradient of the log-likelihood. The gradient of the log-likelihood with respect to the model parameters ($\theta = \{\mathbf{W}, \mathbf{a}, \mathbf{b}\}$) is:

$$\frac{\partial \log p(\mathbf{v})}{\partial \theta} = \mathbb{E}_{p(\mathbf{h}|\mathbf{v})}[\frac{\partial E(\mathbf{v}, \mathbf{h})}{\partial \theta}] - \mathbb{E}_{p(\mathbf{v},\mathbf{h})}[\frac{\partial E(\mathbf{v}, \mathbf{h})}{\partial \theta}],$$

where the first term is the expectation over the data distribution, and the second is over the model distribution. Computing the model expectation is computationally expensive due to the partition function $Z$.

Contrastive Divergence (CD) approximates this gradient by running Gibbs sampling for a small number of steps (k steps, typically $k = 1$), starting from the data. The CD-k algorithm samples a reconstructed visible state $\mathbf{v}'$ from the hidden units derived from the input data, approximating the model distribution. The loss function used in the implementation is the difference in free energy:

$$\text{Loss} = \mathbb{E}[F(\mathbf{v})] - \mathbb{E}[F(\mathbf{v}')].$$

This loss guides the optimization to make the model's distribution closer to the data distribution.

## 3 Implementation and Architecture

The RBM is implemented using PyTorch, with the following key components:

- **Initialization**: The RBM class initializes parameters for visible biases (`v_bias`), hidden biases (`h_bias`), and weights (`W`). The visible layer size (`n_vis`=784) corresponds to the flattened 28x28 MNIST images, and the hidden layer size (`n_hid`=128) is chosen to capture latent features. The number of Gibbs sampling steps is set to $k = 1$.
- **Visible-to-Hidden**: Computes $p(\mathbf{h}|\mathbf{v}) = \sigma(\mathbf{W}^\top \mathbf{v} + \mathbf{b})$, where $\sigma$ is the sigmoid function.
- **Hidden-to-Visible**: Computes $p(\mathbf{v}|\mathbf{h}) = \sigma(\mathbf{W}\mathbf{h} + \mathbf{a})$.
- **Free Energy**: Implements the free energy function as described above.

- **Forward Pass**: Performs Gibbs sampling for $k$ steps to generate reconstructed visible units $\mathbf{v}'$.
- **Training**: Uses the Adam optimizer with a learning rate of 0.0001 and gradient clipping (max norm = 1.0) to stabilize training. The loss is computed as the difference in free energy between the input and reconstructed samples.

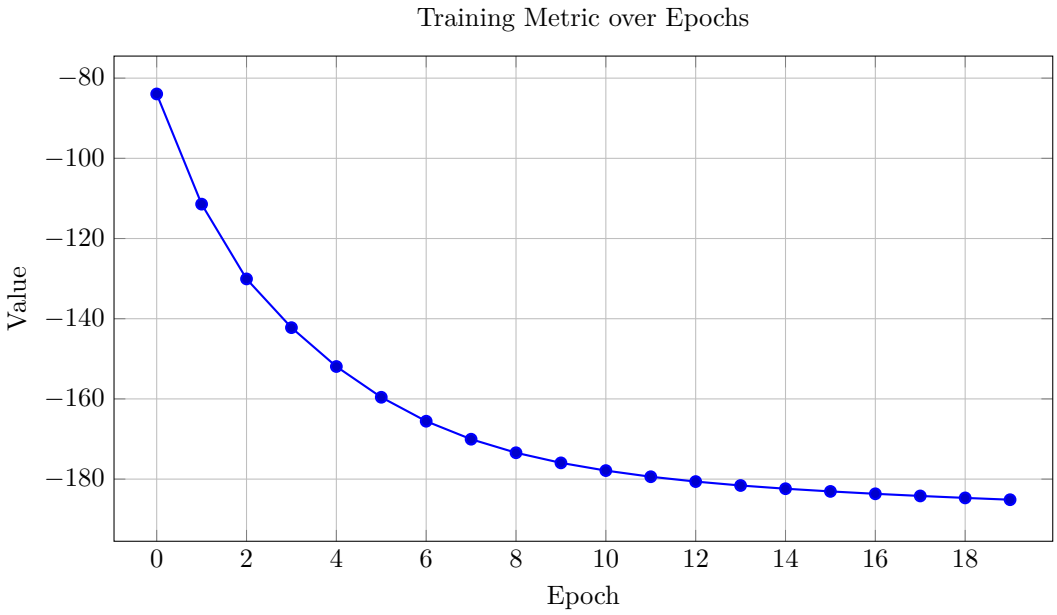
The MNIST dataset is loaded with a binarization transform (`torch.bernoulli`) to convert pixel intensities to binary values (0 or 1), suitable for the RBM's binary visible units. The dataset is processed in batches of 64 images over 20 epochs.

Below is a snippet of the RBM class implementation:

```
class RBM(nn.Module):
    def __init__(self, n_vis, n_hid, k=1):
        super(RBM, self).__init__()
        self.v_bias = nn.Parameter(torch.zeros(n_vis))
        self.h_bias = nn.Parameter(torch.zeros(n_hid))
        self.W = nn.Parameter(torch.randn(n_vis, n_hid) * 0.01)
        self.k = k

    def visible_to_hidden(self, v):
        p_h = torch.sigmoid(F.linear(v, self.W.t(), self.h_bias))
        return p_h

    def hidden_to_visible(self, h):
        p_v = torch.sigmoid(F.linear(h, self.W, self.v_bias))
        return p_v

    def free_energy(self, v):
        v_term = torch.matmul(v, self.v_bias)
        h_term = torch.sum(F.softplus(F.linear(v, self.W.t(),
            self.h_bias)), dim=1)
        return -v_term - h_term

    def forward(self, v):
        v_gibb = v
        for _ in range(self.k):
            h_prob = self.visible_to_hidden(v_gibb)
            h_sample = torch.bernoulli(h_prob)
            v_prob = self.hidden_to_visible(h_sample)
            v_gibb = v_prob
        return v, v_gibb
```

## 4 Training Results

The model was trained for 20 epochs with a batch size of 64, learning rate of 0.0001, and 128 hidden units. The loss values (mean free energy difference per epoch) are reported as follows:




The decreasing loss trend indicates that the model is learning to minimize the free energy difference, suggesting improved reconstruction of the input data over time. The final loss of -185.1377 reflects a stable convergence, though further tuning (e.g., increasing $k$, adjusting the learning rate, or adding more hidden units) could enhance performance.

## 5 Visualizations

To evaluate the model's performance, two figures are provided:

Figure 1 shows the real and generated images at epochs 0, 10, and 19. Initially, the generated images are noisy, but by epoch 19, they begin to resemble MNIST digits, indicating that the RBM is learning meaningful features.

Figure 2 displays a grid of generated samples after 20 epochs, created using the `sample` method with 1000 Gibbs sampling steps. These samples demonstrate the model's generative capability, though some images may still appear noisy due to the simplicity of the model ($k = 1$).

## 6 Analysis and Inferences

The code implements a standard RBM with a single Gibbs sampling step ($k = 1$), which is computationally efficient but may limit the quality of the generated samples. The use of the Adam optimizer and gradient clipping helps stabilize training, preventing large updates that could destabilize the model. The binarization of MNIST images ensures compatibility with the binary visible units of the RBM, though this may discard some grayscale information.

The loss trend suggests successful learning, but the generated images may not be as sharp as those from more complex models (e.g., deep Boltzmann machines or GANs). Increasing $k$, using a higher learning rate, or adding more hidden units could improve performance. The `sample`

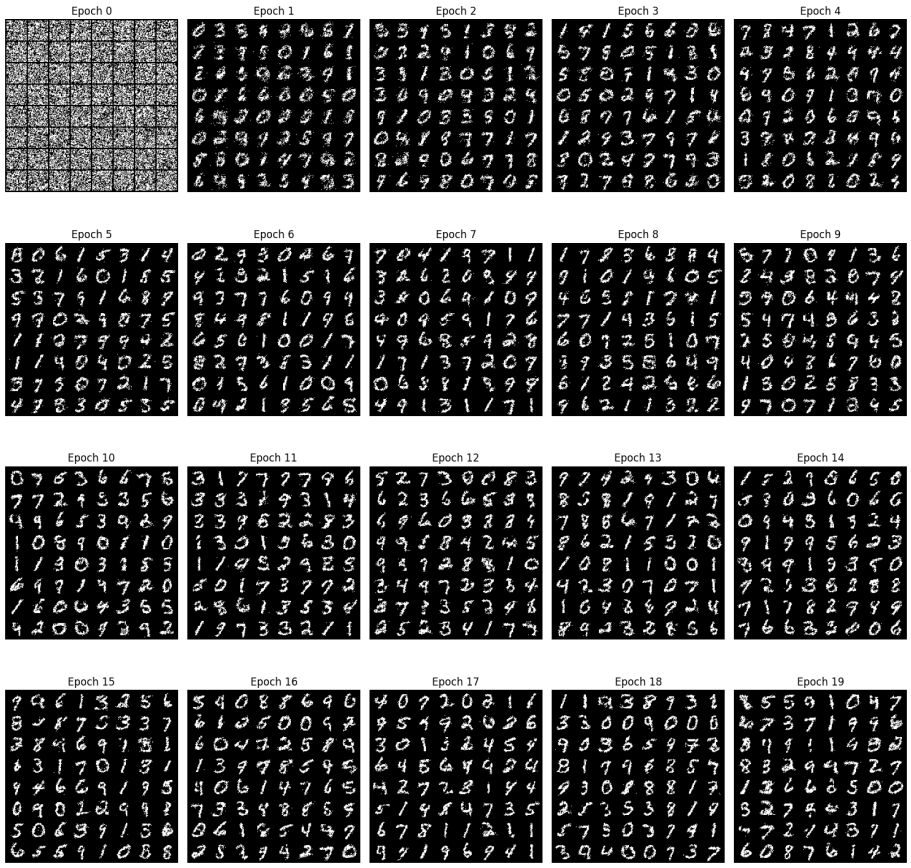



Figure 1: Comparison of real and generated MNIST images at epochs 0, 10, and 19. The generated images show progressive improvement in capturing digit-like structures as training progresses.

method, which uses 1000 Gibbs sampling steps, produces better-quality samples than the forward pass, indicating that longer sampling improves the model's generative quality.

Potential improvements include:

- Increasing $k$ in the forward pass to improve reconstruction during training.
- Experimenting with different learning rates or optimizers (e.g., SGD with momentum).
- Using a larger hidden layer to capture more complex features.
- Applying persistent contrastive divergence for better sampling.

The code assumes GPU availability (CUDA), but falls back to CPU if unavailable, ensuring portability. The visualization function `show_and_save` effectively saves and displays images, aiding in qualitative evaluation.

Figure 2: Grid of generated samples after 20 epochs, showcasing the RBM's ability to produce digit-like images through Gibbs sampling.

## 7 Conclusion

This assignment implemented and trained an RBM on the MNIST dataset, demonstrating its ability to learn and generate digit-like images. The theoretical foundations, including the energy-based formulation and contrastive divergence, were successfully translated into a PyTorch implementation. The training results and visualizations confirm that the model learns meaningful features, though further tuning could enhance the quality of generated samples. This exercise provides a solid foundation for understanding energy-based models and their applications in generative tasks.