Markov chains for gradient estimation in the context of Restricted Boltzmann Machine Learning

Sandra Burgos and Alan García

November 21, 2024

Contents

1 Restricted Boltzmann Machines **2**

2 Data: MNIST **2**

3 Contrastive Divergence (CD) algorithm **3**

4 Persistent Contractive Divergence (PCD) algorithm **7**

5 Results **8**

1 Restricted Boltzmann Machines

The aim of the Restricted Boltzmann Machine (RBM) is to obtain a probabilistic representation of the data, by understanding the patterns that the data follows. An RBM consists of two-layer neural nets:

- The visible layer: consists of m visible units $V = (V_1, ..., V_m) \in \{0, 1\}^m$ to represent observable data.
- The hidden layer: consists of n hidden units $H = (H_1, ..., H_n) \in \{0, 1\}^n$ to capture dependencies between observed variables.

Each connection between the visible and hidden units is assigned a weight, denoted by W_{ij} for all $i \in \{1, ..., n\}$ and $j \in \{1, ..., m\}$. In addition, each unit in the visible and hidden layers has a bias term: b_j and a_j are associated with the jth visible and the ith hidden variables. The nodes are connected to each other across layers, but no two nodes of the same layer are linked - this is the restriction in a restricted Boltzmann machine.

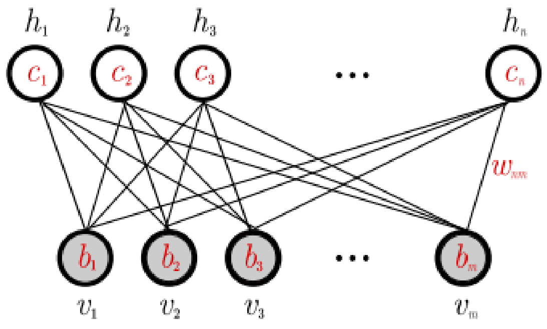


Figure 1: The undirected graph of an RBM with n hidden and m visible variables.

Given random vectors v and h, the energy function of the Restricted Boltzmann Machine (RBM) is written as:

$$E(v, h; \theta) = -\sum_{i=1}^{m}\sum_{j=1}^{n} w_{ij} v_i h_j - \sum_{i=1}^{m} b_i v_i - \sum_{i=1}^{n} a_i h_i$$

where the parameters of the model are $\theta = \{w_{ij}, b_i, c_j\}$. The joint probability of finding v and h in a particular state is given by:

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

where $Z(\theta)$ is the partition function, defined as:

$$Z(\theta) = \sum_{i=1}^{m}\sum_{j=1}^{n} e^{-E(v_i,h_j;\theta)}$$

which is the sum over all possible configurations, and is used to normalize the probability. Notice that higher energy states, are associated with lower probability levels.

2 Data: MNIST

The Modified National Institute of Standards and Technology (MNIST) dataset contains images of handwritten digits (0-9). Each image is 28x28 pixels, which results in a total of 784 pixels.

- The **visible units** will correspond to the 784 pixels in each image. Each unit will hold a binary value indicating whether a pixel is active or not.

Figure 2: MNIST

- The **hidden units** will capture the patterns within the images. For example, it could be: specific combinations of pixels that frequently appear in certain digits or detection of curves. In this project, we used 30 hidden units.

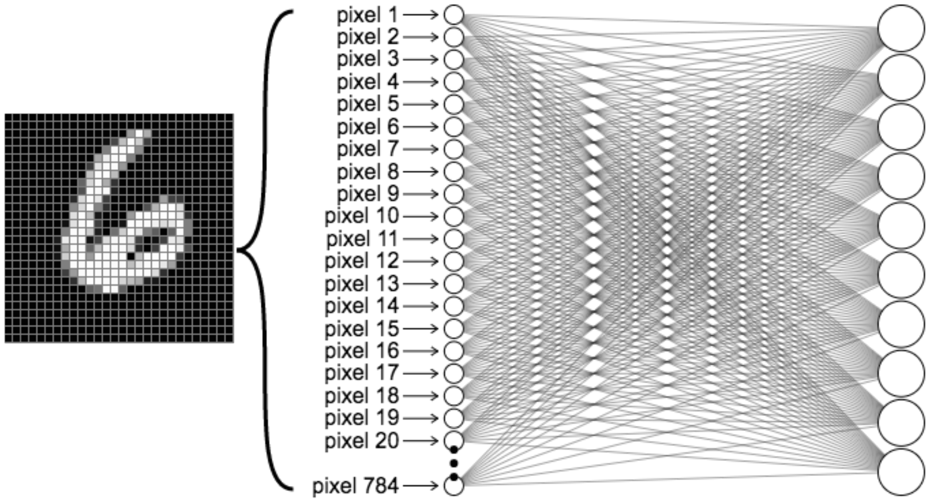


Figure 3: Restricted Boltzmann Machines for the MNIST data

Through training, we will learn the probability distribution of the MNIST data, allowing the model to reconstruct familiar patterns (such as the digits ”3” or ”8”) and generate new, realistic examples that resemble real handwritten digits.

3 Contrastive Divergence (CD) algorithm

The goal is to train the RBM to learn a probability distribution, called the data distribution, from a data set of samples of the distribution, known as the training data. The aim of training is to adjust the parameters $\theta = \{W, b, a\}$ such that the marginal distribution of the visible layer $p(v)$ matches the data distribution represented by the training data set.

The standard approach to solve this problem is based on the maximum likelihood principle. In other words, we want to find the parameters $\theta = \{W, b, a\}$ that maximize the probability of our model producing the data. The maximum likelihood function is written as:

$$L(\theta) = \prod_{k=1}^{m} P(v_k, h_k; \theta) = \prod_{k=1}^{m} \frac{e^{-E(v_k, h_k; \theta)}}{Z(\theta)}.$$

Maximizing the likelihood is equivalent to training the RBM using the logarithm of the likelihood. The log-likelihood is easier to handle because it converts the product into a sum,

$$\log(L(\theta)) = \sum_{k=1}^{m} \log P(v_k, h_k; \theta) = -\sum_{k=1}^{m} E(v_k, h_k; \theta) - \sum_{k=1}^{m} \log Z(\theta) = -f - g = -(f+g).$$

Maximizing the likelihood is equivalent to minimizing the negative log-likelihood, then:

$$L'(\theta) = -\log(L(\theta)) = \sum_{k=1}^{m} E(v_k, h_k; \theta) + \sum_{k=1}^{m} \log Z(\theta) = f + g.$$

The next step to obtain the minimum of this function is to derivate respected to the parameters $\theta = \{a, b, W\}$. Recall that $E(v, h; \theta) = -v^t W h - b^T v - a^T h$. Then, we have

$$\frac{\partial L'(a,b,W)}{\partial W} = \frac{\partial f}{\partial W} + \frac{\partial g}{\partial W}.$$

However, calculating these derivatives exactly leads to some integrals that are intractable. In particular, the partition function $Z(\theta)$ involves summing over all possible configurations of the visible and hidden units, which is computationally expensive and difficult to evaluate exactly, especially for large datasets or complex models.
To address this issue, approximations to the gradient are used. One popular approximation is Contrastive Divergence (CD) although it does not guarantee t¡convergence to the global optimum. For further knowledge on the area, visit [3] and [6]. For the case of $\theta = W$ we have

$$\frac{\partial L'(a,b,W)}{\partial W} = \frac{\partial f}{\partial W} + \frac{\partial g}{\partial W} \cong < vh^t >_{\text{data}} - < vh^t >_{\text{model}},$$

where $< . >$ denotes the expected value. The value $< vh^t >_{\text{data}}$ is called positive phase, and $< vh^t >_{\text{model}}$ is called negative phase.

- The aim of the **positive phase** is to compute the association between the visible layer (input data) and the hidden layer. The probability of activation for a hidden layer h_j is:

  $$P(h_j = 1|v) = \sigma(W_{j.}v + b_j),$$

  where σ denotes the activation funcion (in this case we will use the sigmoid function $\sigma(z) = \frac{1}{1+e^{-z}}$.) In order to obtain the probabilities for the whole hidden layer, we compute:

  $$h = \sigma(Wv + b).$$

  Then, the positive phase is the expectation of vh^t.

- The **negative phase** involves generating data based on the model's parameters (not the actual data) and then computing the association between the visible and hidden layers in this generated data. The steps are the following:

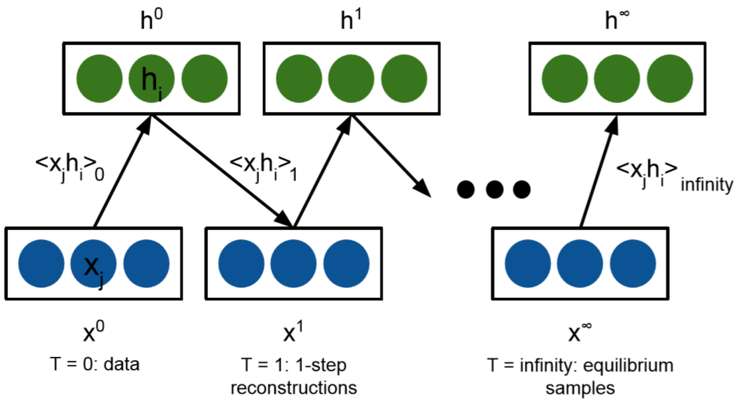
  - Reconstruct visible layers from the hidden activation via Gibbs sampling, which is a Markov Chain Monte Carlo (MCMC) method. The probability of activation for a visible layer v_i is:

    $$P(v_i = 1|h) = \sigma(W_{.i}h + a_i),$$

    then, for the probabilities for the whole hidden layer, we have:

    $$v' = \sigma(Wh + a).$$

  - Reconstruct hidden Layers: $h' = \sigma(Wv' + b)$.
  - Negative association. Obtain and then, $v'h'^T$.



In these process, we can include the Gibbs step k, which alternates between the visible and hidden layers to iteratively refine the state of the system. In the following image, we illustrate this process.

In the training of an RBM, the parameters $\{W, b, a\}$ are updated using the gradients calculated from the contrastive divergence algorithm. If we make this calculus for the different parameters, we obtain:

$$W \leftarrow W + \mu(< vh^t >_{\text{data}} - < v'h'^t >_{\text{model}})$$

$$a \leftarrow a + \mu(< v >_{\text{data}} - < v' >_{\text{model}})$$

$$b \leftarrow b + \mu(< h >_{\text{data}} - < h' >_{\text{model}})$$

where μ represents the learning rate. Then, the pseudo-code is:

```
w,b,a <- inicialize()
for e in (1, ..., E) do:
    for v in training set: # Iterate over the element  in the training set
        # Positive phase
        h_0_p <- W * v + b
        h_0 <- sigmoid(h_0_p)
        # Negative phase
        h_k <- h_0
        for k in (1,..., K) do:   # Gibb sampling with K iterations
            v_k_p <- W * h_k + a
            v_k <- sigmoid(v_k_p)
            h_k_p <- W * v_k + b
            h_k <- sigmoid(h_k_p)
        w <- w + z(v * h_0 - v * h_k.T)
        b <- b + z(v - v_k)
        a <- a + z(h_0 - h_k)
```

We have been able to reconstruct the following digits.

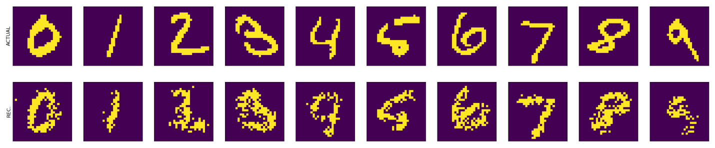


Figure 4: Reconstruction of digits using CD

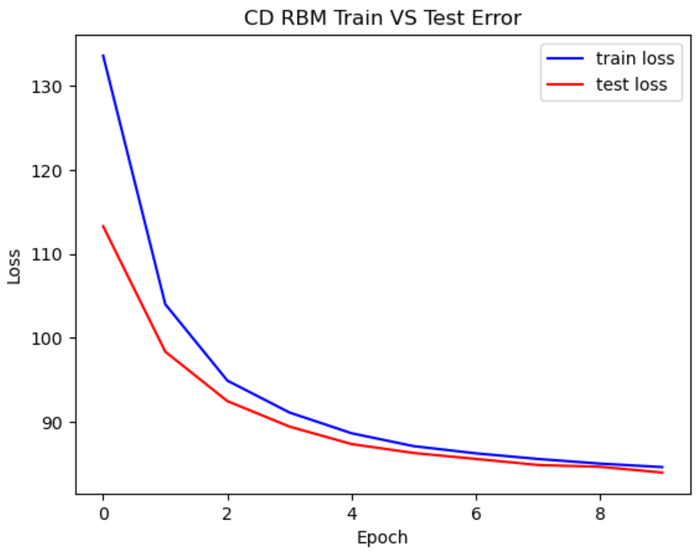


Figure 5: Train VS Test Error

It is observed a decrease in both cases. We have obtained very similar errors for the train and test loss, this means that the model is generalizing well.

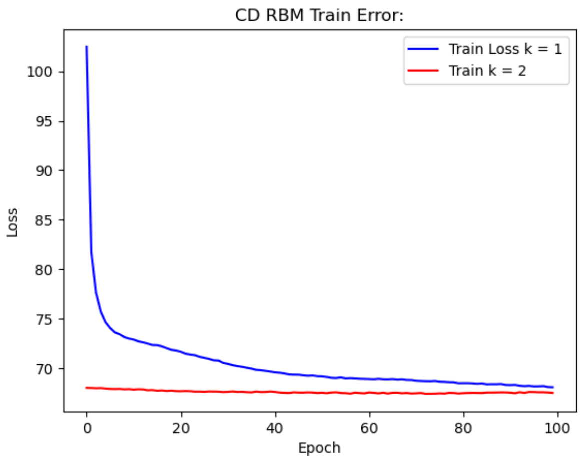


Figure 6: Errors for different Gibbs iterations

For $K = 1$ only one iteration of Gibbs is performed for each sample. This approximation is faster and computationally cheaper. However, when $k = 2$ is used, the visible and hidden layers are updated twice and improves the approximation of the visible layers very quick, although it is computationally expensive.

4 Persistent Contractive Divergence (PCD) algorithm

Due to the approximation error, CD learning does not necessarily lead to a maximum likelihood estimate of the model. Therefore, other alternatives can be used such as Persistent CD, Tempered Transitions or Parallel Tempering. In this work, we will cover the Persistent CD.

The PCD algorithm does not start the Gibbs sampling at a random state or at a data point, as CD does. Instead, it uses a persistent chain that is not reinitialized each time the parameters are changed. Therefore, PCD avoids the issue of forgetting previous samples, as seen in CD, by using the persistent chain which leads to a more accurate estimation of the true gradient. The rest of the algorithm is maintained the same as CD.

```

w,b,a <- inicialization()
# Inicialization of persistent chain
v_persistent <- random.matrix(batch_size, len(input))
for e in (1, ..., E) do:
    for v in training set: # Iterate over the element  in the training set

        # Gibbsampling of v_persistent
        v_pers = v_persistent
        for i in range(self.k):
            h_pers = forward(v_pers)
            v_pers = backward(h)
        h_p_pers_ = forward_probs(v_pers)

        # Positive phase of v (input)
        h_p = forward_probs(v)

        possitive_association = v.T * h_p
        negative_association = v_pers.T * h_p_pers_

        w <- w + z(possitive_association - negative_association)
        b <- b + z(v - v_pers)
        a <- a + z(h_0 - h_p_pers_)
```

Using this algorithm, we have obtained the following reconstruction:

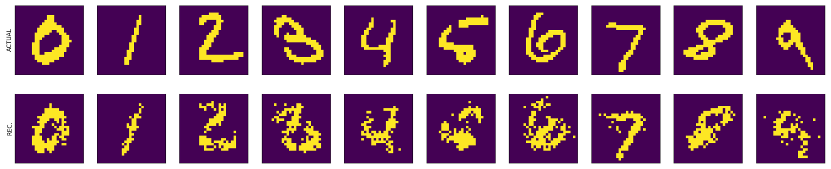


Figure 7: Reconstruction of digits using PCD

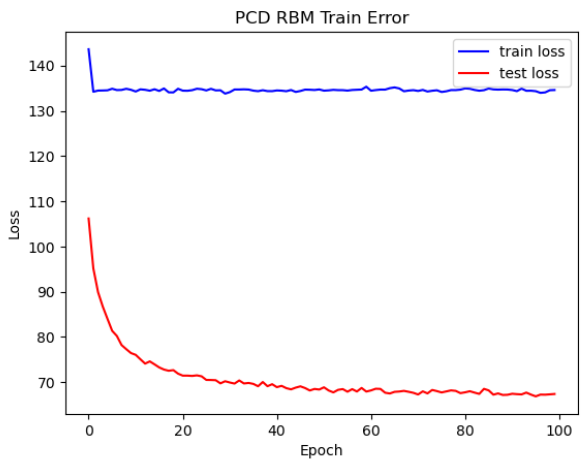


Figure 8: Train VS Test Error in PCD

The results from the PCD algorithm, differ from the CD. In this case, the test error is much smaller than the train error. This result does not make sense. To be honest, we were not able to find error in our algorithm. We tried to modify the hyperparameters, such as the learning rate, the number of hidden units...., but we did not improve the learning error.

5 Results

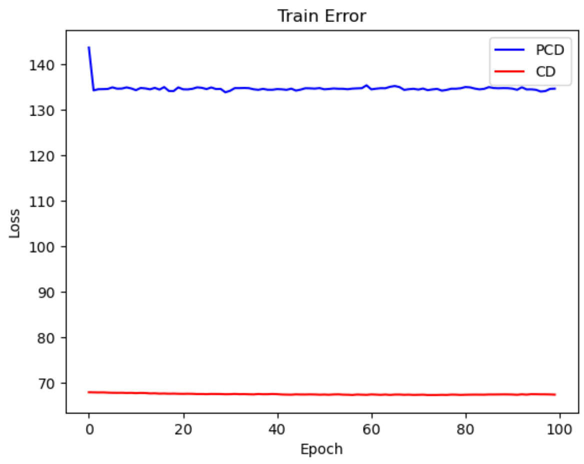


Figure 9: CD k = 2 VS PCD algorithm - Training

For the **training error,** we see a clear difference on using PCD or CD $k = 2$, CD has much smaller error. This can happen because the persistent chains are not directly tied to the data at each iteration, the samples generated are less directly reflective of the training data, leading to slightly higher training error.

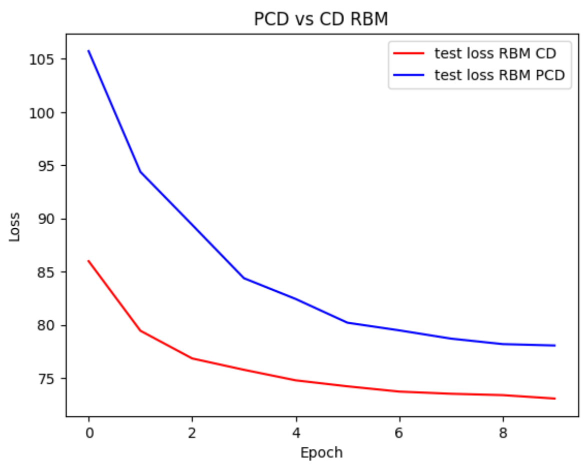


Figure 10: CD VS PCD algorithm - Test

For the **test errors**, however, we see similarities with the error, although in the PCD we have high error. Notice that we have only run it for 10 epochs and we already see some stability for the errors. This could be due to some errors on the performance of the model. However, we have tried with different hyperparameter, such as the learning rate and the number of hidden layers, but no better solution could be found.

Feature	Contrastive Divergence (CD)	Persistent Contrastive Divergence (PCD)
Inicialization	New start for each training example	Persistent across the training examples
Accurary	Less accurate due to insufficient sampling	More accurate due to better sampling
Computational Efficiency	More efficient	Less efficient

In conclusion, despite the difficulties during development, the implementation of the Contrastive Divergence and Persistent Contrastive Divergence algorithms for training Restricted Boltzmann Machines has been successfully completed. However, the final results do not reflect the true comparison between both algorithms, likely due to some error or bug in the implementation that we have not been able to correct despite our efforts.

References

[1] Asja Fischer and Christian Igel. Training restricted boltzmann machines: An introduction. pattern recognition,. 47(1):25–39, 2014.

[2] Sven Behnke Hannes Schulz, Andreas Müller. Investigating convergence of restricted boltzmann machine learning. *Workshop on Deep Learning and Unsupervised Feature Learning*, 2010.

[3] Sven Behnke Hannes Schulz, Andreas Müller. Investigating convergence of restricted boltzmann machine learning. *Workshop on Deep Learning and Unsupervised Feature Learning*, 2010.

[4] Geoffrey E Hinton. Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800, 2002.

[5] Tijmen Tieleman. Training restricted boltzmann machines using approximations to the likelihood gradient. *In Proceedings of the 25th international conference on Machine learning*, pages 1064–1071. ACM, 2008.

[6] Oliver Woodford. Notes on constrastive divergence.