Energy Based Learning
Classification task using Restricted Boltzmann Machines

Igor Atzeni, Eugenio Sebastiani

November 07, 2013

Contents

1 Introduction

An important field of machine learning is the study of the Energy-based models (EBMs). Basically, they associate energies to features probabilities such that high values correspond to high improbability and low values correspond low probabilities, hence, they model a probability distribution over variables. Boltzmann Machines (BMs) are energy-based probabilistic models that define a probability distribution through an energy function as previous. During the last years the restricted form of BMs, the Restricted Boltzmann Machines (RBMs), are used to deal with many problems and it has been shown that RBMs are a powerful generative models able in feature extraction and in the construction of deep artificial neural networks. In our project we want to show that RBMs are also performant in classification tasks with good results. To do this, we have studied the ClassificationRBMs (ClassRBMs), a variant of the RBM adapted to the classification setting. In particular, we have implemented three different type of learning: the *generative learning*, the *discriminative learning* and the *hybrid learning*. We will compare these three different approaches and we will analyze their property. Also we will compare the result obtained with the ClassRBMs with the result obtained classifying the same dataset with a Support Vector Machine (SVM).

2 Energy Based Models

One of the main purpose of the machine learning is to encode dependencies between variables. Energy-Based Models (EBMs) capture this dependencies by associating a scalar energy (a measure of compatibility) to each configuration of the variables. In this case, inference consists in setting the value of observed variables and finding values of the remaining variables that minimize the energy, while learning consists in finding an energy function that associates low energies to correct values of the remaining variables, and higher energies to incorrect values. Also in EBMs a central role is played by the loss functional, this functional is minimized during learning and it is used to measure the quality of the available energy functions. A distinction should be made between the energy function and the loss functional, the first one is minimized by the inference process, and the second one is minimized by the learning process. To better understand the EBMs we can consider a simple model with two variables $\mathbf{x}$ and $\mathbf{y}$. The $\mathbf{x}$ could be a vector representation of one object and the $\mathbf{y}$ is a discrete variable that represents the category of the object. The model is viewed as an energy function which measures the "goodness" (or "badness") of each possible configuration of $\mathbf{x}$ and $\mathbf{y}$. The best energy function is searched within a family of energy functions ε parameterized by $\mathbf{W}$

$$\varepsilon = \{\mathrm{E}(\mathbf{W}, \mathbf{y}, \mathbf{x}) : \mathbf{W} \in \mathcal{W}\} \tag{1}$$

The output number can be interpreted as the degree of compatibility between the values of $\mathbf{x}$ and $\mathbf{y}$ and as already mentioned, by convention is used that small energy value correspond to highly compatible configurations of the variables, while large energy values correspond to highly incompatible configuration of the variables. In other words the model produce the value $\mathbf{y}^*$, chosen from a set $\mathcal{Y}$, for which $\mathrm{E}(\mathbf{y}, \mathbf{x})$ is the smallest:

$$\mathbf{y}^* = \arg\min_{\mathbf{y} \in \mathcal{Y}} \mathrm{E}(\mathbf{y}, \mathbf{x}) \tag{2}$$

To train the model for prediction, classification or decision-making, we use a set of training samples $S = \{(X^{(i)}, Y^{(i)}) : i = 1....P\}$, where $X^{(i)}$ is the input for the i-th training sample and the $Y^{(i)}$ is the corresponding desired answer. Given this set we can use the $loss functional$, denoted by $L(\mathrm{E}, S)$, in order to find the best energy function in the family ε.

$$\mathbf{W}^* = \min_{\mathbf{W} \in \mathcal{W}} L(\mathbf{W}, S) \tag{3}$$

Given the $\mathbf{W}^*$ we can use the energy function for the inference process, in general finding the best $\mathbf{y}$ may not be simple, for example in the situation in which the set $\mathcal{Y}$ may be too large to make an exhaustive search. In these cases is important to find a specific strategy, called the inference procedure, to find the best $\mathbf{y}$. The best inference procedure to use, often depends on the internal structure. In other words different situations may require different optimization procedures like: gradient-based optimization algorithm, continuous optimization methods, quadratic programming, non-linear optimization methods, discrete optimization methods and graph matching. Moreover in many case, exact optimization is impractical, and one must resort to approximated methods.

3 Boltzmann Machine

The Boltzmann Machine (BM) is a network of symmetrically connected and stochastic neural-like units that is well suited for constraint satisfaction tasks with a large number of "weak" constraints. Intuitively, the purpose of these networks is to model the statistical behavior of same part of the world and typical application are pattern classification, generation of plausible patterns or reconstruction of partial patterns. The BMs are composed by primitive computing elements called *units* and every unit is connected to each other by bidirectional links. Also the units of a BM can be divided in two sets, the *visible units* $\mathbf{v}$ that are used for the data clamping and the *hidden units* $\mathbf{h}$ that are used like latent variables that allow the Boltzmann machine to model distributions over visible state vectors.

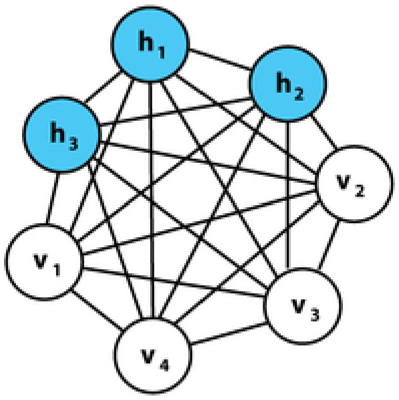

Figure 1: Example of a BM.

In every moment one unit i can be in one of two states, *on* or *off* and it adopts the state s_i as a probabilistic function of the states of its neighboring units j and the weights w_{ij} on its links to them:

$$p(s_i = 1) = \frac{1}{1 + \exp(-b_i + \sum_j s_j w_{ij})} \tag{4}$$

where b_i is the bias term of the unit i. The weight on a link represents a weak pairwise constraint between two hypotheses, they are symmetric, having the same strength in both directions. Hence, positive weight indicates that the two hypotheses tend to support one another; if one is currently accepted, accepting the other should be more likely. Conversely, a negative weight suggests that the two hypotheses should not both be accepted. The weights and the biases of a BM define an *energy function* over every global *configuration* of the network, this energy function is defined as:

$$\mathrm{E}(\mathbf{v},\mathbf{h}) = -\sum_i b_i s_i^{(\mathbf{v},\mathbf{h})} - \sum_{i<j} s_i^{(\mathbf{v},\mathbf{h})} s_j^{(\mathbf{v},\mathbf{h})} w_{ij} \tag{5}$$

It can be shown that if the units are chosen in random way and continuously updated using the eq. (4) the network will eventually reach their Boltzmann distribution (also called equilibrium or stationary distribution). In this distribution the probability of any configuration $(\mathbf{v},\mathbf{h})$ is taken only by the energy of that configuration relative to the energies of all other possible configurations of the network:

$$p(\mathbf{v},\mathbf{h}) = \frac{\exp(-\mathrm{E}(\mathbf{v},\mathbf{h}))}{\sum_{\mathbf{u},\mathbf{g}} \exp(-\mathrm{E}(\mathbf{u},\mathbf{g}))} \tag{6}$$

and easily can be show that the probability to find the network in a certain configuration over its visible units $\mathbf{v}$ is given by the equation:

$$p(\mathbf{v}) = \frac{\sum_{\mathbf{h}} \exp(-\mathrm{E}(\mathbf{v},\mathbf{h}))}{\sum_{\mathbf{u},\mathbf{g}} \exp(-\mathrm{E}(\mathbf{u},\mathbf{g}))} \tag{7}$$

Given the model described so far and one training set of state vectors, the training consists on finding weights and biases that define a Boltzmann distribution in which the training vectors have high probability. Let $P^{+}(\mathbf{v})$ be the distribution over the training set and let $P^{-}(\mathbf{v})$ be the Boltzmann distribution after its marginalization over the hidden units, can be considered the Kullback-Leibler measure KL to evaluate the distance between the two distribution and learning amounts to minimize this distance:

$$\mathrm{KL} = \sum_{\mathbf{v}} P^{+}(\mathbf{v}) \ln\left(\frac{P^{+}(\mathbf{v})}{P^{-}(\mathbf{v})}\right) \tag{8}$$

Since the eq. (8) depends on the weights and the biases, the model can be improved by modifying the w_{ij} and the b_i so as to reduce the KL and a simple *gradient descent* strategy can be used to do this. In other words every w_{ij} is updated by subtracting the partial derivative of KL with respect to the weight, and surprisingly this derivative is simply given by:

$$\frac{\partial \mathrm{KL}}{\partial w_{ij}} = -(< p_{ij}^{+} > - < p_{ij}^{-} >) \tag{9}$$

where $< p_{ij}^{+} >$ is the average probability of two units, the i^{th} and the j^{th}, both being in the *on* state when data vectors from $P^{+}(\mathbf{v})$ are clamped on the states of the visible units measured at equilibrium, and $< p_{ij}^{-} >$ is the corresponding probability when the environmental input is not present and the network is running freely at equilibrium. Commonly the computation of $< p_{ij}^{+} >$ is called *positive phase* and the computation of $< p_{ij}^{-} >$ is called *negative phase*. Following the gradient descent strategy, once these two quantities have been computed, the learning process simply consist in increasing the value of w_{ij} in the opposite direction of the gradient:

$$\Delta w_{ij} = \gamma(< p_{ij}^{+} > - < p_{ij}^{-} >) \tag{10}$$

where $\gamma > 0$ is the learning rate. The updating of the biases is similar and given by:

$$\Delta b_{i} = \gamma(< p_{i}^{+} > - < p_{i}^{-} >) \tag{11}$$

A surprising feature of this method is that it uses only locally available information. The change in a weight depends only on the behavior of the two units it connects, even though the change optimizes a global measure, and the best value for each weight depends on the values of all the other weights. This learning algorithm is very simple but it has also big weak points. Firstly the network reach the equilibrium only after very long time also in the situation in which an heuristic is used to accelerate the learning, and second, the learning signal is very noisy because it is the difference of two approximated expectations. This problem makes the algorithm impracticable in the situation in which there are large networks with many units.

4 Restricted Boltzmann Machine

As their name implies, the Restricted Boltzmann Machines (RBMs) are a variant of the BM, they are BMs with a restricted architecture. Indeed RBM consists of a layer of visible units and a layer of hidden units without connections between units of the same layer (no connection visible-visible or hidden-hidden), and usually every units of one layer is connected to all units

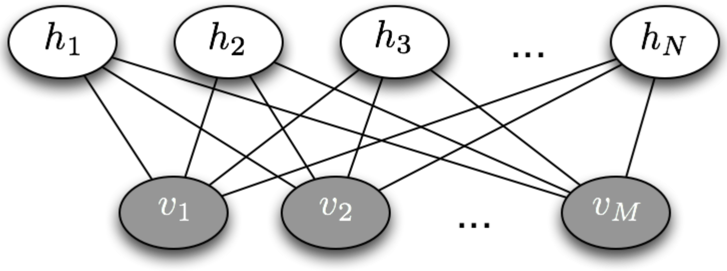

Figure 2: Structure of an RBM

in the other layer. With this restricted architecture we can see the energy function associated at the RBM as:

$$\mathrm{E}(\mathbf{v},\mathbf{h}) = -\sum_i a_i v_i - \sum_j b_j h_j - \sum_{i,j} v_i h_j w_{ij} \tag{12}$$

where v_i and h_j are the binary states of visible unit i and hidden unit j, a_i and b_j are their biases and w_{ij} is the weight between them. Also with this energy function and no connections between nodes of the same layer, the probability that one hidden node is on and the probability that one visible node is on are given by the following equations:

$$p(h_j = 1|\mathbf{v}) = \frac{1}{1 + \exp(-b_j + \sum_i v_i w_{ij})} \tag{13}$$

$$p(v_i = 1|\mathbf{h}) = \frac{1}{1 + \exp(-a_i + \sum_j h_j w_{ij})} \tag{14}$$

The restriction simplify the learning and allows for more efficient algorithms than are available for the general class of Boltzmann machines. Indeed, with this architecture the probability of generating a visible vector is proportional to the product of the probabilities that the visible vector would be generated by each of the hidden units acting alone.

An RBM is therefore a “product of expert” (PoE) with one expert per hidden unit and this observation leads us to consider the optimization of one different objective function called *Contrastive Divergence* (CD) instead of just minimizing the Kullback-Leibler measure shown in eq. (8).

$$\mathrm{CD}_n = \left(\sum_{\mathbf{v}} P^+(\mathbf{v}) \ln \left(\frac{P^+(\mathbf{v})}{P^-(\mathbf{v})} \right) - \sum_{\mathbf{v}} P^n(\mathbf{v}) \ln \left(\frac{P^n(\mathbf{v})}{P^-(\mathbf{v})} \right) \right) \tag{15}$$

where $P^n(\mathbf{v})$ is the distribution over the one-step reconstruction of the data vectors generated by n full step of Gibbs sampling procedure, that consists in updating visible and hidden units in chain, alternating between updating all the visible units in parallel and updating all the hidden units in parallel. As done with the BM it is possible to use the gradient descent strategy for minimizing the CD function and so improving the model by modifying $\Theta = (\mathbf{W}, \mathbf{b}, \mathbf{a})$. To do this, it can be shown that the derivatives of the CD with respect to w_{ij}, a_i and b_j are approximated by the following simple equations:

$$\frac{\partial \mathrm{CD}}{\partial w_{ij}} = -(< p_{ij}^+ > - < p_{ij}^T >) \tag{16}$$

$$\frac{\partial \mathrm{CD}}{\partial da_i} = -(< p_i^+ > - < p_i^T >) \tag{17}$$

$$\frac{\partial \mathrm{CD}}{\partial b_j} = -(< p_j^+ > - < p_j^T >) \tag{18}$$

So the learning rules become:

$$\Delta w_{ij} = \gamma(< p_{ij}^+ > - < p_{ij}^T >) \tag{19}$$

$$\Delta a_i = \gamma(< p_i^+ > - < p_i^T >) \tag{20}$$

$$\Delta b_j = \gamma(< p_j^+ > - < p_j^T >) \tag{21}$$

where $< p_{ij}^+ >$ is given by eq. (13) and $< p_{ij}^T >$ is given by T steps of the Gibbs sampling procedure. The base step is very simple:

1. Starting with a data vector on the visible units, the hidden units are all updated in parallel using eq. (13).

2. Update all visible units in parallel to get a "reconstruction" using eq. (14).

3. Update all hidden units again.

CD learning uses this step for T times, computes the p_{ij}, repeat the procedure for the entire dataset and computes the average to get $< p_{ij}^T >$.

5 Classification using RBM

The Classification Restricted Boltzmann Machines (ClassRBMs), as suggested by the name, are a particular type of RBM developed for the classification task. The ClassRBM models the joint distribution of an input $\mathbf{x} = (x_1, ..., x_D)$ and target $y \in \{1, ..., C\}$ using a hidden layer of binary stochastic units $\mathbf{h} = (h_1, ..., h_N)$ and in this case the energy function of the model is:

$$\mathrm{E}(y, \mathbf{x}, \mathbf{h}) = -\mathbf{h}^T\mathbf{W}\mathbf{x} - \mathbf{b}^T\mathbf{x} - \mathbf{c}^T\mathbf{h} - \mathbf{d}^T\mathbf{e}_y - \mathbf{h}^T\mathbf{U}\mathbf{e}_y \tag{22}$$

with parameters $\Theta = (\mathbf{W}, \mathbf{b}, \mathbf{c}, \mathbf{d}, \mathbf{U})$, where $\mathbf{W}$ is the weights matrix between $\mathbf{x}$ and $\mathbf{h}$, $\mathbf{U}$ is the weights matrix between $\mathbf{e}_y$ and $\mathbf{h}$, and $\mathbf{b}$, $\mathbf{c}$ and $\mathbf{d}$ are respectively the biases of $\mathbf{x}$, $\mathbf{e}_y$ and $\mathbf{h}$. Lastly the $\mathbf{e}_y = (1_{i=y})_{i=1}^C$ is the "one out of C" representation of y. From the energy function, the joint probability of y, $\mathbf{x}$ and $\mathbf{h}$ is given by:

$$p(y, \mathbf{x}, \mathbf{h}) = \frac{\exp(-\mathrm{E}(y, \mathbf{x}, \mathbf{h}))}{Z} \tag{23}$$

where Z is a normalization constant. This probability is equivalent to the one saw in eq. (6) splitting $\mathbf{v}$ in $\mathbf{x}$ and $\mathbf{e}_y$. As shown in the previous paragraph, it is intractable, but fortunately we can use the Gibbs sampling procedure again to overcome the problem. Indeed, all the conditional distributions that are required for the Gibbs sampling are easily computed. In particular it can be shown that:

$$p(h_j = 1|y, \mathbf{x}) = \mathrm{sigm}(c_j + u_{jy} + \sum_i w_{ji}x_i) \tag{24}$$

$$p(x_i = 1|\mathbf{h}) = \mathrm{sigm}(b_i + \sum_j w_{ji}h_j) \tag{25}$$

$$p(y|\mathbf{h}) = \frac{\exp(d_y + \sum_j u_{jy}h_j)}{\sum_{y^*} \exp(d_{y^*} + \sum_j U_{jy^*}h_j)} \tag{26}$$

It is also possible to compute $p(y|\mathbf{x})$ that will be used later:

$$\begin{aligned} p(y|\mathbf{x}) = & \frac{\exp(d_y + \sum_j \text{softplus}(c_j + u_{jy} + \sum_i w_{ji} x_i))}{\sum_{y^* \in (1,\ldots,C)} \exp(d_{y^*} + \sum_j \text{softplus}(c_j + u_{jy^*} + \sum_i w_{ji} x_i))} \\ = & \frac{\exp(-\mathrm{F}(y,\mathbf{x}))}{\sum_{y^* \in (1,\ldots,C)} \exp(-\mathrm{F}(y,\mathbf{x}))} \end{aligned} \tag{27}$$

where $\mathrm{F}(y,\mathbf{x})$ is referred to as the free energy.

To train a ClassRBM to solve a particular classification problem, one can simply define an objective function to minimize for all example in the set Θ, as was done in the RBM. It is possible to choose different objective functions, in particular there are three approach that are discussed below:

1. Generative Training Objective.
2. Discriminative Training Objective.
3. Hybrid Training Objective.

Generative Training Objective Choosing a Generative Training Objective, it is possible to train a model that is able to solve the classification problem, but that is also able to generate the dataset used for the training. To train this type of model, given that we have a model which defines a value for the joint probability $p(y^{(t)}, \mathbf{x}^{(t)})$, a natural choice for the training objective is the negative log-likelihood of this joint probability (this is equivalent to minimizing the Kullback-Leibler measure shown in eq. (8), as we did before):

$$L_{gen}(D_{train}) = -\sum_{t=1}^{|D_{train}|} \log p(y^{(t)}, \mathbf{x}^{(t)}) \tag{28}$$

This is the most popular training objective for RBMs but, as seen before, it is affected by a defect, computing the $p(y^{(t)}, \mathbf{x}^{(t)})$ for some example $(y^{(t)}, \mathbf{x}^{(t)})$ is generally intractable, as is computing the $\log p(y^{(t)}, \mathbf{x}^{(t)}))$ and its gradient with respect to any ClassRBM parameter $\theta = (\mathbf{W}, \mathbf{b}, \mathbf{c}, \mathbf{d}, \mathbf{U})$:

$$\frac{\partial \log p(y^{(t)}, \mathbf{x}^{(t)})}{\partial \theta} = \mathrm{E}_{\mathbf{h}|y^{(t)},\mathbf{x}^{(t)}} \frac{\partial}{\partial \theta} \mathrm{E}(y^{(t)}, \mathbf{x}^{(t)}, \mathbf{h}) + \mathrm{E}_{y,\mathbf{x},\mathbf{h}} \frac{\partial}{\partial \theta} \mathrm{E}(y, \mathbf{x}, \mathbf{h}) \tag{29}$$

where the E is the expected value. In particular the second expectation is intractable but, it is possible to use the Contrastive Divergence algorithm, showed in the fourth paragraph, to overcome this problem. In other words the Generative ClassRBM is a RBM that use the CD algorithm and in which architecture the visible units are divided in the input **x** and class **y**.

Discriminative Training Objective If the interest is to make a model that is able to solve the classification problem only and you are not interested to the generative capability, it is possible to choice a Discriminative Training Objective. For this aim is useful to see that the generative training objective can be decomposed as follow:

$$\begin{aligned} L_{gen}(D_{train}) &= -\sum_{t=1}^{|D_{train}|} (\log p(y^{(t)}|\mathbf{x}^{(t)}) + \log p(\mathbf{x}^{(t)})) \\ &= -\sum_{t=1}^{|D_{train}|} (\log p(y^{(t)}|\mathbf{x}^{(t)})) - \sum_{t=1}^{|D_{train}|} (\log p(\mathbf{x}^{(t)})) \end{aligned} \tag{30}$$

The right hand of the equation suggest that using the generative training objective ClassRBM will dedicate some of its capacity at modeling the marginal distribution of the input only, giving at the model the generative capability. Since we are in a supervised learning setting, if we are only interested to obtain a good prediction of the target given the input, is possible to ignore the unsupervised part of the generative objective and focus on the supervised part. So we can consider as Discriminative Training Objective only the left hand of the eq. (26):

$$L_{disc}(D_{train}) = -\sum_{t=1}^{|D_{train}|} \log p(y^{(t)}|\mathbf{x}^{(t)}) \tag{31}$$

The most important advantage using this training objective is that is possible to compute its gradient with respect to the ClassRBM's parameters, in particular we obtain that for $\Theta = (\mathbf{c}, \mathbf{U}, \mathbf{W})$:

$$\begin{aligned}\frac{\partial \log p(y^{(t)}|\mathbf{x}^{(t)})}{\partial \Theta} &= \sum_j \mathrm{sigm}(o_{y^{(t)}j}(\mathbf{x}^{(t)}))\frac{\partial o_{y^{(t)}j}(\mathbf{x}^{(t)}))}{\partial \Theta} \\ &- \sum_{j,y^*} \mathrm{sigm}(o_{y^*j}(\mathbf{x}^{(t)}))p(y^*|\mathbf{x}^{(t)})\frac{\partial o_{y^*j}(\mathbf{x}^{(t)})}{\partial \Theta}\end{aligned} \tag{32}$$

where $o_{yj}(\mathbf{x}) = c_j + \sum_k w_{jk}x_k + u_{jy}$, the gradient whit respect to the ClassRBM's parameters $\mathbf{d}$ is:

$$\frac{\partial \log\ p(y^{(t)}|\mathbf{x}^{(t)})}{\partial \mathbf{d}} = 1_{y=y^{(t)}} - p(y|\mathbf{x}^{(t)}), \quad y \in (1, ..., C) \tag{33}$$

Notice that the gradient with respect to $\mathbf{b}$ is 0, since the input biases are not involved in the computation of $p(y|\mathbf{x})$. Given the gradient with respect the ClassRBM's parameters is possible to update these parameters using a gradient descent strategy.

Hybrid Training Objective The generative and discriminative approaches have different properties that can influence the choice of use one or the other. For example the generative approach yields a model that can reach more rapidly their best generalization performance, but when the model is misspecified, the discriminative training objective makes a model that reach better performance for sufficiently large data. Moreover for models from the general exponential family, parameters estimates based on the generative training objective have smaller variance than discriminative estimates, but if the model is misspecified, generative estimates will asymptotically yield models with worse performances. This properties suggest that the generative models as being more regularized than discriminative models. So in the classification task that we discuss, adding the generative training objective to the discriminative training objective is a way to regularize this second one and to adapt the amount of regularization to the problem at hand, is possible to use for the training process an Hybrid Training Objective:

$$L_{hybrid}(D_{train}) = L_{disc}(D_{train}) - \alpha L_{gen}(D_{train}) \tag{34}$$

where the weight α can be adjusted to obtain the best performance.

6 Experiment

Initially we have implemented the training algorithm of a RBM in Matlab language using one step of Contrastive Divergence (CD-1). As a dataset we have used a toy example as in [5]. It was composed of 10 small images representing simple geometric patterns, and to train the RBM we have cloned it many times randomly in order to have a dataset of 1000 elements. Once we have trained the RBM we conduct two experiment: in one we start from a random configuration, in the other we start from a corrupt image of the dataset. From the firs, after performing the Gibbs Sampling until convergence, we get random images with some noise, from the second we get exactly the correct images before the corruption.

This was a good starting point to implement the Classification RBM (ClassRBM). The improvement was to split the visible units in input units and output units, and the correspondent weights and biases, we have added the classification units as the output, and the discriminative learning.

The output of a classificator should be a unit that can assume k different states, if there are k different classes. The common units in an RBM are only binary, and the probability of turning they on is a logistic sigmoid function of its total input x:

$$p = \sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x + e^0} \tag{35}$$

This comes from the Boltzmann distribution where the energy contributed by the units is $-x$, if it is on, and 0 if it is off. This distribution can be used even if there are k different states, where the contribution in energy $-x_j$ from the units depend on the state j.

$$p_j = \frac{e^{x_j}}{\sum_{i=1}^{k} e^{x_i}} \tag{36}$$

This kind of units are called “softmax”. We can simulate this behavior using k binary units by requiring that only one can be turned on at the time. In this way the learning rule remain the same as the standard binary units; the only change is the calculation of the probability and then the relative Gibbs sampling.

The initial values of the weights (then the values of $\mathbf{W}$ and $\mathbf{U}$) are set to random small values with zero mean in a range between -0.1 and 0.1. The biases are initialized to zero.

To speed up the learning we have adopted some techniques, explained in the following.

We have split the entire dataset in batches and updated after each one the weights and the bias, so that at the end of an epoch the progress in the learning is greater.

To keep the values of the weights small we can use the weight-decay. It consists of adding another task: minimizing the "L2" function, that is, the half of the square of the weights times a constant c called weight-cost:

$$|\mathbf{W}|^2 = c \sum_{w_{ij}} \frac{w_{ij}^2}{2} \tag{37}$$

In the gradient descent method it is simple to achieve this task by adding an extra term: the derivative of this penalty function, i.e. the weights times the weight-cost. There are other benefits to use weight-decay: it improves generalization to new data by reducing overfitting to the training data, avoiding that some units became always *on* or always *off* because of the too large weights. Also, small weights lead to a better mixing rate of the alternating Gibbs Markov chain; for the contrastive divergence with one step it is a really good improvement, because it ignores the derivatives coming from the other steps and the faster is the convergence, the smaller are the ignored derivatives.

The momentum is used to decrease the probability of remaining stuck in a local minimum and it smooths the path in the objective function made by the gradient descent, increasing the learning speed. This technique is inspired from physics and it is based on the idea to give to the gradient descent some inertia. In this analogy, we can see the process of minimization as a ball rolling down from a mountain, the greater is the momentum the larger is the ball. Hence, if it remains stuck in local optima it has the inertia that allows it to keep moving until it escapes. The recipe of Hinton is to start with a momentum of 0.5 and once the learning slow down it is the right time to change to 0.9, and this works very well in our case.

The learning rate decreases with the increasing of the epochs according to the eq. (38) performing a kind of simulated annealing. Also this is done on the attempt to avoid local minima.

$$\gamma(t) = \frac{t_0}{1 + \frac{t}{\tau}} \tag{38}$$

Another technic that we have used is an averaging over the last 10 epochs before stoping the learning over the weights and the bias. Some times it accelerate learning and provide better result.

To avoid overfitting we measure the classification error on the validation dataset. When it starts to grow, is a signal that the classification is specializing too much on the training data and is loosing generality.

As a term of comparison we have trained a Support Vector Machine (SVM) using a library called svmlib. We have used a linear kernel and a Radial basis function kernel (Gaussian).

The generative phase is done using only one step of Contrastive Divergence, it is very fast and allow us to use a grater number of hidden units. Instead, if we use the discriminative learning we have to reduce the number of hidden units because otherwise the values involved in the calculations became too large to be handled by matlab.

In this experiment we have used the MNIST dataset. That is a dataset composed of images representing hand written numbers. At each image is associated the correct label. The MINST is divided in a training dataset of 60 000 examples and a test set of 10 000 examples. From the training set we extract the validation set composed of 10 000 examples.

7 Results

We have performed three type of experiment using the different type of learning. As we expected the Discriminative learning is much better than the Generative one in the classification task: where the discriminative capabilities between different input is more important then the learning of the exact probability distribution of the input. The best result is given when we use both technics, suggesting that they are in some way complementary.

We have done many tries using different values of the number of hidden units, the learning rate γ, the weight cost and the weight of the generative learning in the hybrid configuration α. For simplicity other parameters like the size of the batch and the initial and final momentum are kept fixed. This cross validation on parameter selection can be done in a finer manner using much more computational power.

In the learning process we have recorded three valuation measurements: the reconstruction error, the total free energy and the classification error. The first measure the squared error between the data and the reconstruction after a Gibbs sampling. It give us an suggestion if the learning is going well or not. In our case it fall rapidly at the beginning and then more slowly, as it is supposed to do. The total free energy is used to compute the probabilities $p(\mathbf{y}|\mathbf{x})$ in the discriminative learning with the correct label $(\mathbf{y})$, than it is a measure that tells us if the discriminative learning is performing well. It has to decrees like the classification error. The last evaluation is the squared error between the predicted label and the correct one in the validation set. It is the most important because it is the error in the task that we are trying to solve and it sum up the two previous measurement.

Table 1 show the classification error on the test set in the tree type of learning plus the parameters that have brought us to that result and the classification error using SVM with different kernels.

In the following are shown the valuation measurements recorded during the learning in the three kind of learning: only generative, only discriminative and hybrid. The trend is as we expect, the reconstruction error and the free energy fall down until they reach a constant value, even the classification error on the validation set stabilizes on a certain value that is comparable to the classification error on the test set.

Model	Objective	Error
ClassRBM	Generative (γ=0.5 to 0.01, 1500 hidden units)	8.54
	Discriminative(γ=0.75 to 0.01, 800 hidden units)	3.61
	Hybrid	1.90
SVM	Linear	6.02
SVM	RBF(C=2, g=0.1)	4.31

Table 1: Errors on the test set

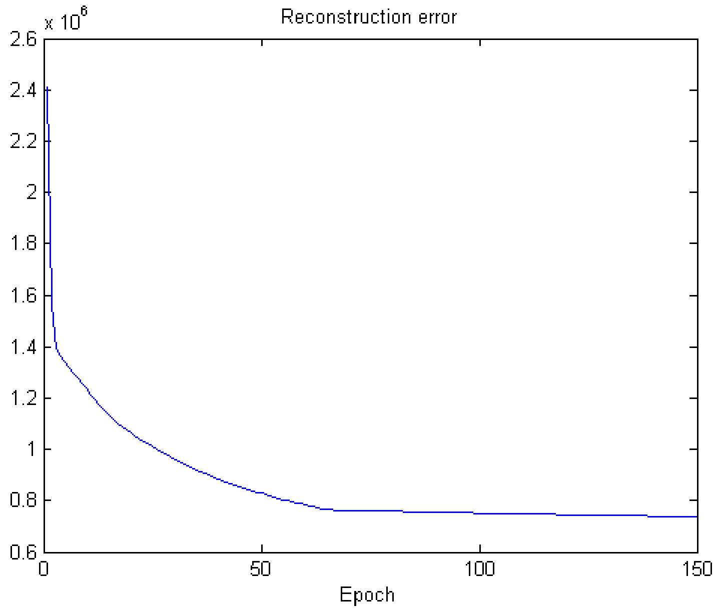

Figure 3: Reconstruction error during learning in generative training.

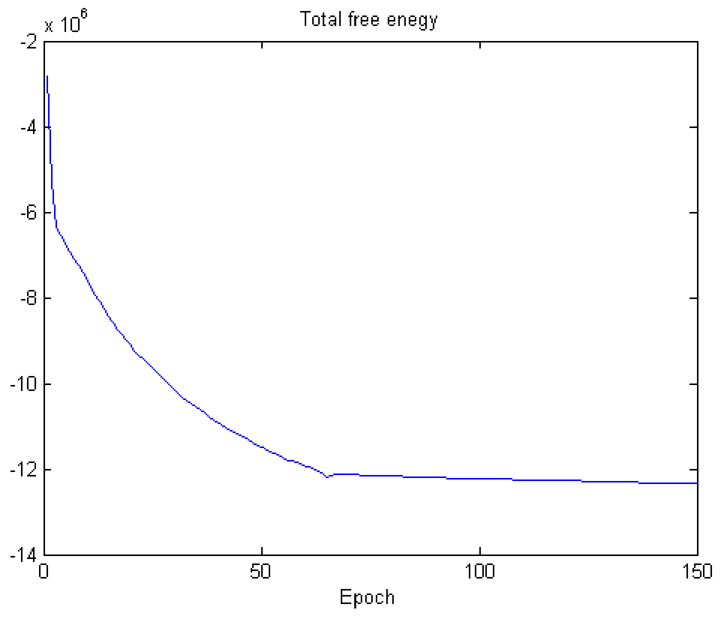

Figure 4: Free energy during learning in generative training.

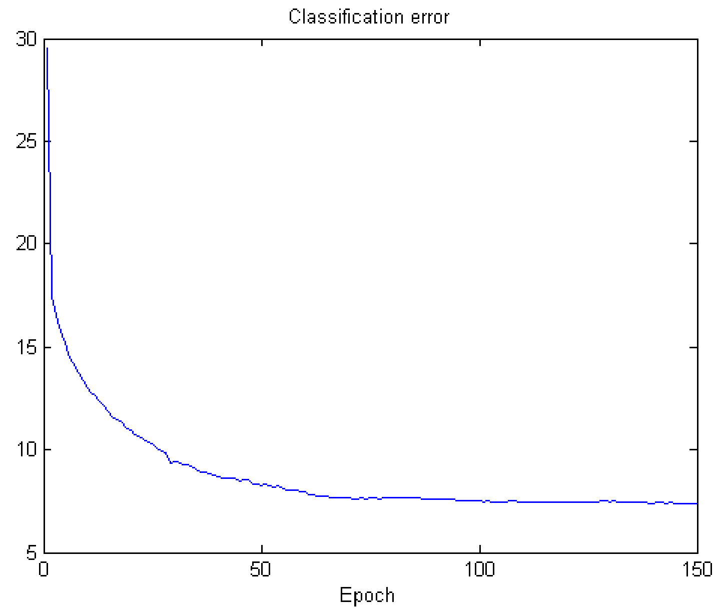

Figure 5: Classification error during learning in generative training.

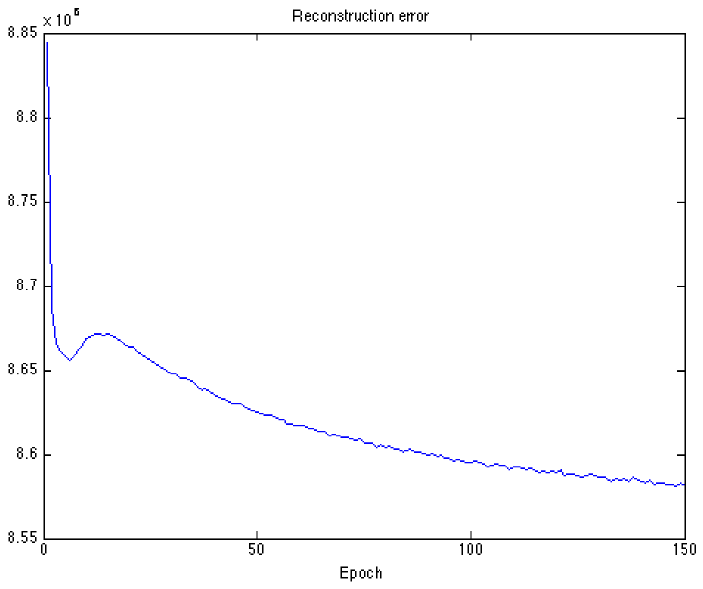

Figure 6: Reconstruction error during learning in discriminative training.

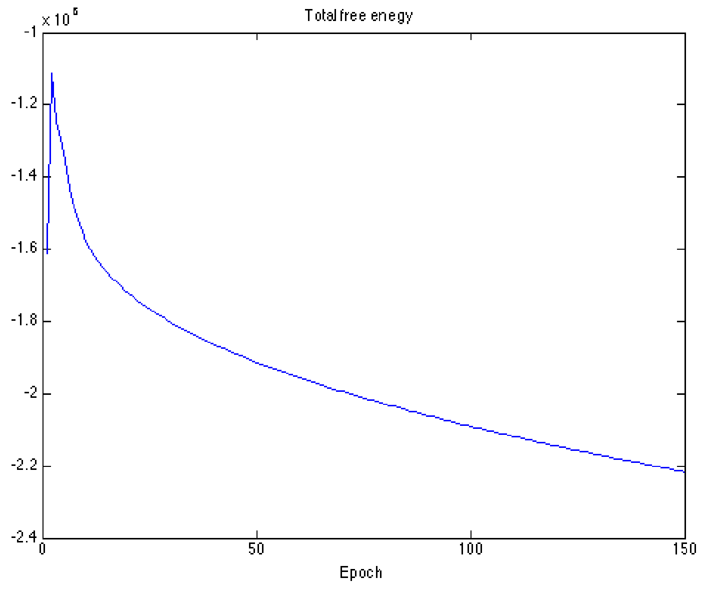

Figure 7: Free energy during learning in discriminative training.

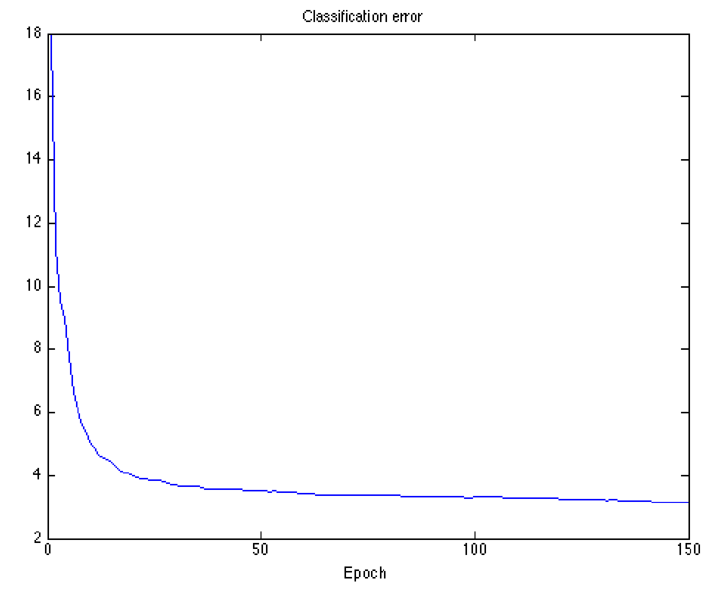

Figure 8: Classification error during learning in discriminative training.

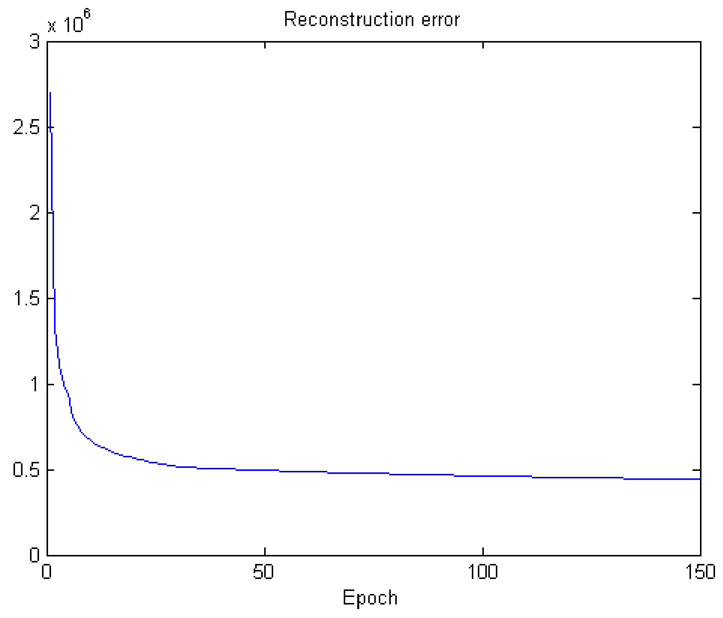

Figure 9: Reconstruction error during learning in hybrid training.

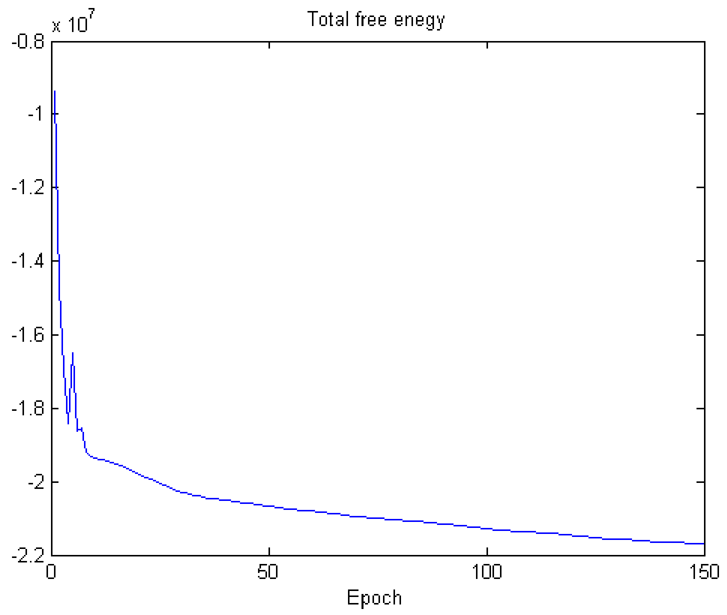

Figure 10: Free energy during learning in hybrid training.

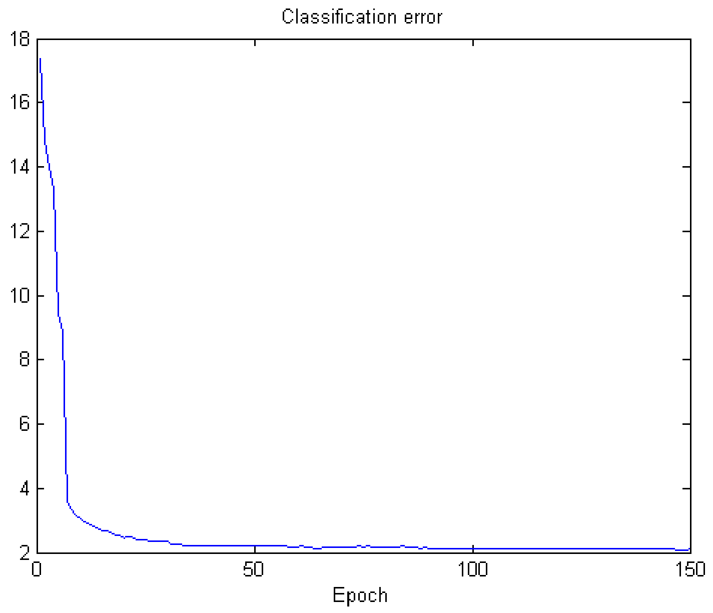

Figure 11: Classification error during learning in hybrid training.

8 Conclusion

The RBMs are used mostly as features extractors and also in deep learning structures. Furthermore, to solve the classification problems, the hidden units of a RBMs have been used as input for some standard discriminative method. In our work we have used it as a stand-alone discriminative classifier. The classification errors on the test set that we have obtained are comparable to the errors obtained by a SVM, a common algorithm used to solve a classification problem. The error using a pure generative learning is quite high but the discriminative give us a very good result, even better than the one using a SVM with RBF kernel. The best performance on our work are achieved using an hybrid configuration. It is due to the fact that the discriminative training do the most part of the job and the generative act as a form of regularization.

In the generative and hybrid mode, even if this is a discriminative task, the model maintain the generative capabilities, that may be useful for same different applications.

References

[1] Yann LeCun, Sumit Chopra, Raia Hadsell, MarcAurelio Ranzato, and Fu Jie Huang *A Tutorial on Energy-Based Learning.* MIT Press', 2006.

[2] DH Ackley, GE Hinton, TJ Sejnowski. *A learning algorithm for Boltzmann machines'* - Cognitive science, 1985 - Elsevier.

[3] Ludovic Arnold, Sbastien Rebecchi, Sylvain Chevallier, Hlne Paugam-Moisy. *An Introduction to Deep Learning.* 2012.

[4] Hugo Larochelle, Michael Mandel, Razvan Pascanu, and Yoshua Bengio. *Learning Algorithms for the Classification Restricted Boltzmann Machine.* Journal of Machine Learning Research 13, pages 643-669. 2012.

[5] Gilles Louppe, Collaborative Filtering. Master thesis at University of Lige. 2009-2010.

[6] Carreira-Perpignan, M. A. and Hinton. G. E. On Contrastive Divergence Learning. In: Artificial Intelligence and Statistics, 2005, Barbados

[7] Hinton, G.E Training Products of Experts by Maximizing Contrastive Divergence. Technical Report: GCNU TR1999-001