Energy Based Models
Classification using Restricted Boltzmann Machine

Neural Networks Project

Atzeni Igor, Sebastiani Eugenio

SAPIENZA
UNIVERSITÀ DI ROMA

27 novembre 2013

Introduction

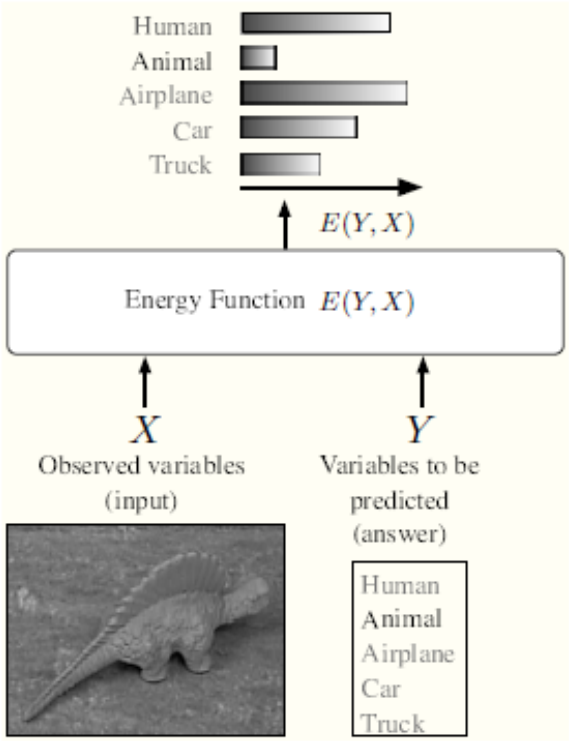
In machine learning classification is the problem of identifying to which of a set of categories a new observation belongs, on the basis of a training set of data containing observations whose category membership is known.
In our project we have implemented a Restricted Boltzmann Machine to deal this problem.

Energy Based Models 1/2

Energy Based Models capture dependencies between variables by associating a scalar energy to each configuration of the variables.
The model can be viewed as an energy function which measure the goodness (or badness) of each possible configuration.
By convention small energy values correspond to highly compatible configurations.

Energy Based Models 2/2

X could be pixels of an input, and Y a discrete label describing the object in the image. Given X, the model produces the answer Y that minimizes the energy E.



Inference

Given the value of the input $\mathbf{x}$, the inference procedure produces the value $\mathbf{y}$, chosen from a set $\mathcal{Y}$, for which $\mathrm{E}(\mathbf{y}, \mathbf{x})$ is the smallest:

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

Training 1/2

Training an EBM consists in finding an energy function that produces the best $\mathbf{y}$ for one $\mathbf{x}$.
The search for the best energy function is performed within a family of energy functions ε indexed by a parameter $\mathbf{W}$:

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

To train the model we have a set of training samples
$S = \{(\mathbf{x}^i, \mathbf{y}^i) : i = 1....P\}$, where $\mathbf{x}^i$ is the input for the i-th training sample, and $\mathbf{y}^i$ is the corresponding desired answer.

Training 2/2

In order to find the best energy function in the family ε we need a way to assess the quality of each energy function.
This quality measure is called the **loss function** $L(W, S)$ and the learning problem is simply to find the W that minimize that loss:

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

BM Architecture 1/2

Boltzmann Machines are networks of symmetrically connected and stochastic neural-like units.
Are composed by **VISIBLE** and **HIDDEN** units and every unit can be in one of two states: **ON** or **OFF**.
In particular every unit 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:

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

BM Architecture 2/2

The weights and the biases of a BM define an *energy function* over every global *configuration* of the network:

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

Choosing the values of the units in random way and continuously updating the network, it will reach the Boltzmann distribution (or equilibrium).
When the network is in the Boltzmann 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:

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

Training algorithm 1/4

For the BMs, training consists on finding weight and biases that define a Boltzmann distribution in which the training vector has high probability.
Given:

- $P^+(\mathbf{v})$, the distribution over the training set
- $P^-(\mathbf{v})$, the Boltzmann distribution after its marginalization over the hidden units

we can consider the Kulmann-Leibler measure to evaluate the distance between this two distribution:

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

Training algorithm 2/4

Since the KL considered depends on weights and biases, the model can be improved by modifying the w_{ij} and the b_i as to reduce the KL. To do this a simple *gradient descent* strategy can be used.

$$w_{ij} = w_{ij} - \gamma(\frac{\partial \mathrm{KL}}{\partial w_{ij}})$$

$$b_i = b_i - \gamma(\frac{\partial \mathrm{KL}}{\partial b_i})$$

Training algorithm 3/4

Surprisingly the partial derivative of KL with respect the weights and the biases are simple given by:

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

$$\frac{\partial \mathrm{KL}}{\partial b_{i}} = (< p_{i}^{+} > - < p_{i}^{-} >)$$

Where:

- $< p_{ij}^{+} >$ is the average probability of two units 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.
- $< p_{ij}^{-} >$ 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*.

Training algorithm 4/4

TrainingAlgorithm – PositivePhase

1. Clamp a data vector on the visible units of the Boltzmann machine.
2. Update the hidden units in random order.
3. Once the Boltzmann machine has reached its equilibrium distribution, sample state vectors and record p_{ij}.
4. Repeat steps 1, 2 and 3 for the entire dataset. Average to get $< p_{ij}^{+} >$.

TrainingAlgorithm – NegativePhase

1. Initialize the Boltzmann machine with a random state.
2. Update visible and hidden units in random order.
3. Once the Boltzmann machine has reached its equilibrium distribution, sample state vectors and record p_{ij} .
4. Repeat steps 1, 2 and 3 many times. Average to get $< p_{ij}^{-} >$.

Pro & Con

PRO:

- BM has a very simple learning algorithm

CON:

- The model can take a very long time to reach equilibrium. The time required to settle to equilibrium actually grows exponentially with the number of units.

RBMs Architecture 1/2

Restricted Boltzmann Machines (RBMs) are a variant of BMs with a restricted architecture.
RBM consists in a layer of visible units and a layer of hidden units, without connection between units of the same layer.

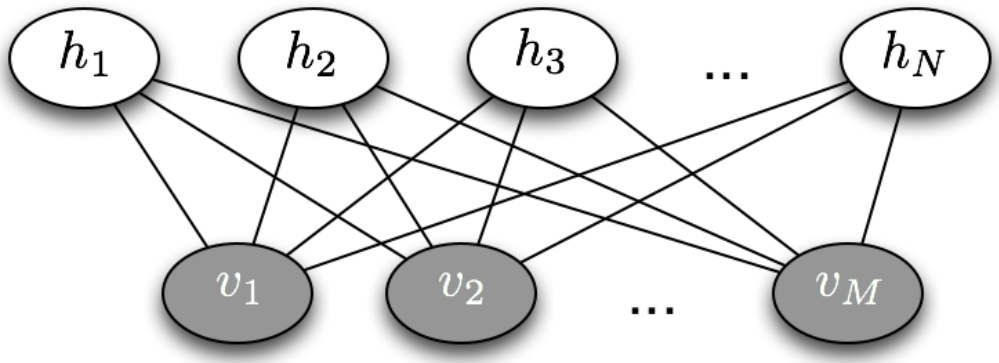


Figura : Structure of an RBM

RBMs Architecture 2/2

With this restricted architecture the energy function associated at the model becomes:

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

- 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;
- w_{ij} is the weight between the units i and j.

And also we have:

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

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

RBMs Training Algorithm - Contrastive Divergence 1/2

With this restricted architecture can be used one different objective function, the *Contrastive Divergence* (CD):

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

where $P^n(\mathbf{v})$ is the distribution over the one-step reconstructions 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.

RBMs Training Algorithm - Contrastive Divergence 2/2

Gradient descend strategy is still used to minimize the CD function and so improve the model by modifying $\Theta = (\mathbf{W}, \mathbf{b}, \mathbf{a})$.

$$\frac{\partial \mathrm{CD}}{\partial w_{ij}} = -(< p_{ij}^{+} > - < p_{ij}^{T} >) \quad ; \quad \Delta w_{ij} = \gamma(< p_{ij}^{+} > - < p_{ij}^{T} >)$$

$$\frac{\partial \mathrm{CD}}{\partial da_{i}} = -(< p_{i}^{+} > - < p_{i}^{T} >) \quad ; \quad \Delta a_{i} = \gamma(< p_{i}^{+} > - < p_{i}^{T} >)$$

$$\frac{\partial \mathrm{CD}}{\partial b_{j}} = -(< p_{j}^{+} > - < p_{j}^{T} >) \quad ; \quad \Delta b_{j} = \gamma(< p_{j}^{+} > - < p_{j}^{T} >)$$

where $< p_{ij}^{T} >$ is given by T steps of the Gibbs sampling procedure.

RBMs Training Algorithm - Gibbs Sampling

The base step of Gibbs sampling is very simple:

1. Starting with a data vector on the visible units, the hidden units are all updated in parallel.
2. Update all of the visible units in parallel to get a reconstruction .
3. Update all of the hidden units again.

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

ClassRBMs Architecture 1/2

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:

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

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.

ClassRBMs Architecture 2/2

To train a ClassRBM we still use the Gibbs sampling and all the conditional distribution required are easily computed:

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

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

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

We can also compute:

$$p(y|\mathbf{x}) = \frac{\exp(-\mathrm{F}(y, \mathbf{x}))}{\sum_{y^* \in (1,\ldots,C)} \exp(-\mathrm{F}(y, \mathbf{x}))}$$

Training Algorithms

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

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

Generative Training Objective

A natural choice for the training objective is the negative log-likelihood of the joint probability $p(y^{(t)}, \mathbf{x}^{(t)})$:

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

It can be shown that minimizing this training objective is equivalent to minimizing the Contrastive Divergence.
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 $\mathbf{x}$ and class $\mathbf{y}$.

Discriminative Training Objective 1/2

The Generative Training Objective can be decomposed as follow:

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

If we are only interested to obtain a good prediction of the target given the input, it is possible to ignore the unsupervised part of the generative objective (the right hand) and focus on the supervised part (the left hand), losing the generative capability of the model. So we can consider as Discriminative Training Objective:

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

Discriminative Training Objective 2/2

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 $\Theta = (\mathbf{c}, \mathbf{U}, \mathbf{W})$:

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

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

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

Hybrid Training Objective

The generative and discriminative approaches have different properties. In particular we can see that the generative model is more regularized than discriminative model. 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.To adapt the amount of regularization to the problem at hand, is possible to use for the training process an Hybrid Training Objective:

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

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

The Experiment

We have implemented an RBM in Matlab language and tested on a classification problem. The RBM can be trained with three different type of learning:

- generative training using CD1,
- discriminative training,
- hybrid training.

To test the model given by the learning a prediction procedure on a test set is done.
Comparing the predicted labels to the correct ones we get the errors.

Softmax units

The output units must be different from the other, because it has to model a probability distribution over all the classes.
The softmax units is suitable to our purpose.
Supposing that we have k classes the probability that the j-th units is *on* is:

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

where x is the input of the units.
With this probability distribution we can observe that the probability that one of the units is *on* is one.

The Dataset

MNIST dataset: a collection of images representing hand written numbers from 0 to 9.

- The training set is composed of 60 000 examples.
- The test set is composed of 10 000 examples.
- The training set was divided in 500 batches.
- 10 000 examples was reserved to a validation set to test the evolution of the training.

Speed up the learning

To reach the minimum of the objective function we use the gradient descent method enriched with some techniques:

- using batches
- momentum
- weight-decay
- simulated annealing

Each technique is parameterized, then we had to perform a cross validation on parameter selection to obtain goods values.
With this parameter it was essential to determine the number of hidden units, moreover because of the limited computational calculus the we had available we cannot reach a perfect number. This step is really computational expensive.

Batches

- Using batches leads to a really important speed up in learning.
- Instead of update the weights and the bias after each epoch, they are updated after each batch is computed.
- It is important to find a good trade-off between the number of batches and the size of them.
- A big number of batches means many updates every epoch but if the size is too small the update is too specified in that batch and it don't generalize.

Momentum

- The momentum is used to decrease the probability of remaining stuck in local minima.
- Also smooths the path of the objective function increasing the learning speed.
- It is like to add inertia to the learning:
  if it runs into a local minimum with inertia it has the energy to go ahead.
- In our implementation we change the value after 5 epochs because of the different update magnitude between the beginning and the end.

Weight-decay

- It is used to keep the values of the weights small.
- It consists of adding another task: minimizing the "L2" function:

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

- it is simple to achieve this task by adding an extra term: the weights times the weight-cost.
- It improves generalization to new data by reducing overfitting to the training data,
- leads to a better mixing rate of the alternating Gibbs Markov chain.

Simulated annealing and Averaging

- In Simulated annealing the learning rate decreases with the increasing of the epochs according to:

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

- It accelerate the learning, helps convergence and avoiding local minima.

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

A term of comparison

- In order to compare our RBM implementation with other classification algorithms we use a Support Vector Machine.
- In particular we used the "svmlib" implementation in matlab.
- As a kernel we used the linear and a Radial basis function performing a parameter selection of C and g.

Results

This table summarizes the results the we have obtained comparing with the SVM performance:

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

Considerations about the experiments

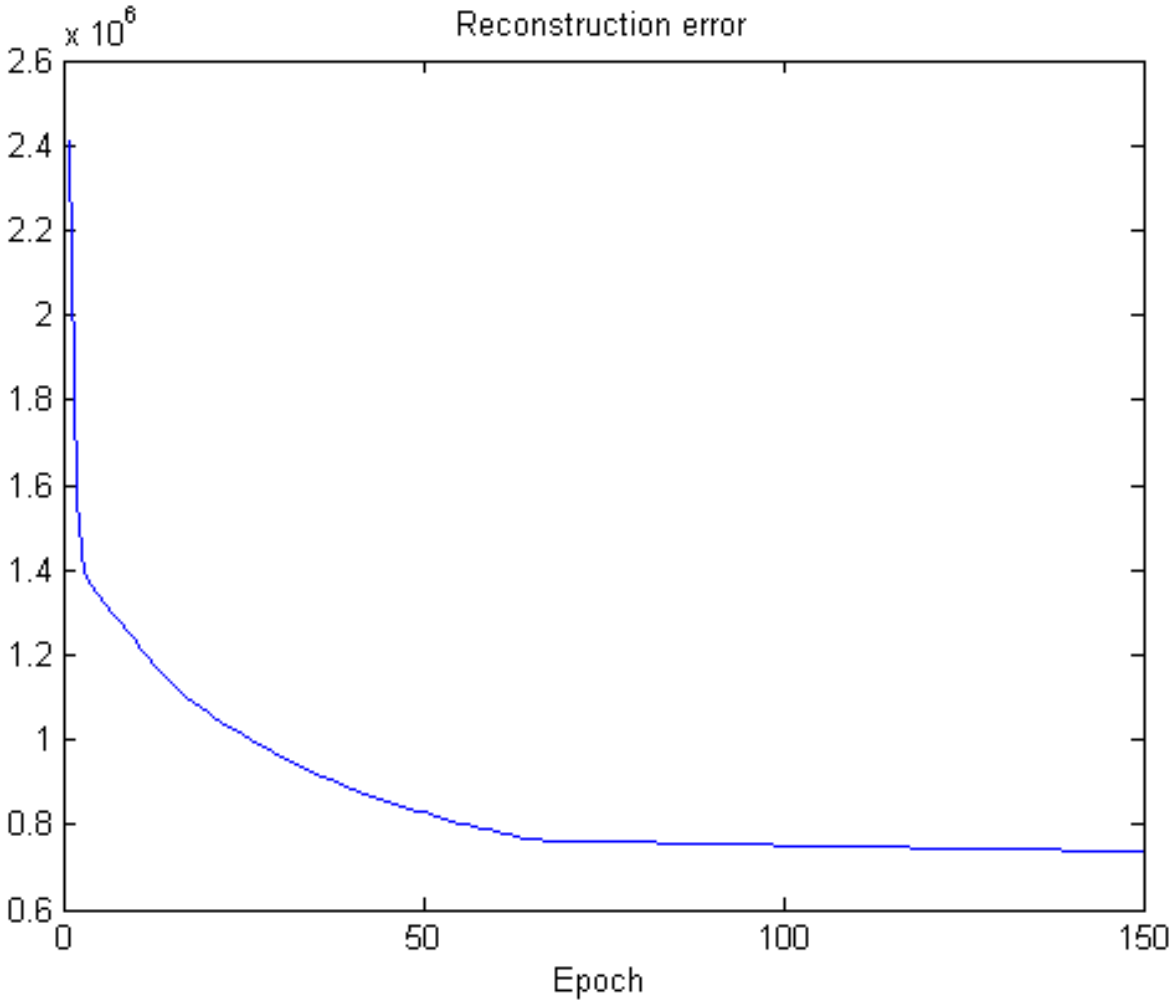
As we expected the Discriminative learning is much better than the Generative one in the classification task: where the discriminative capabilities between different input is more important then the learning of the exact probability distribution of the input.
The best result is given when we use both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

Valuation Measurements

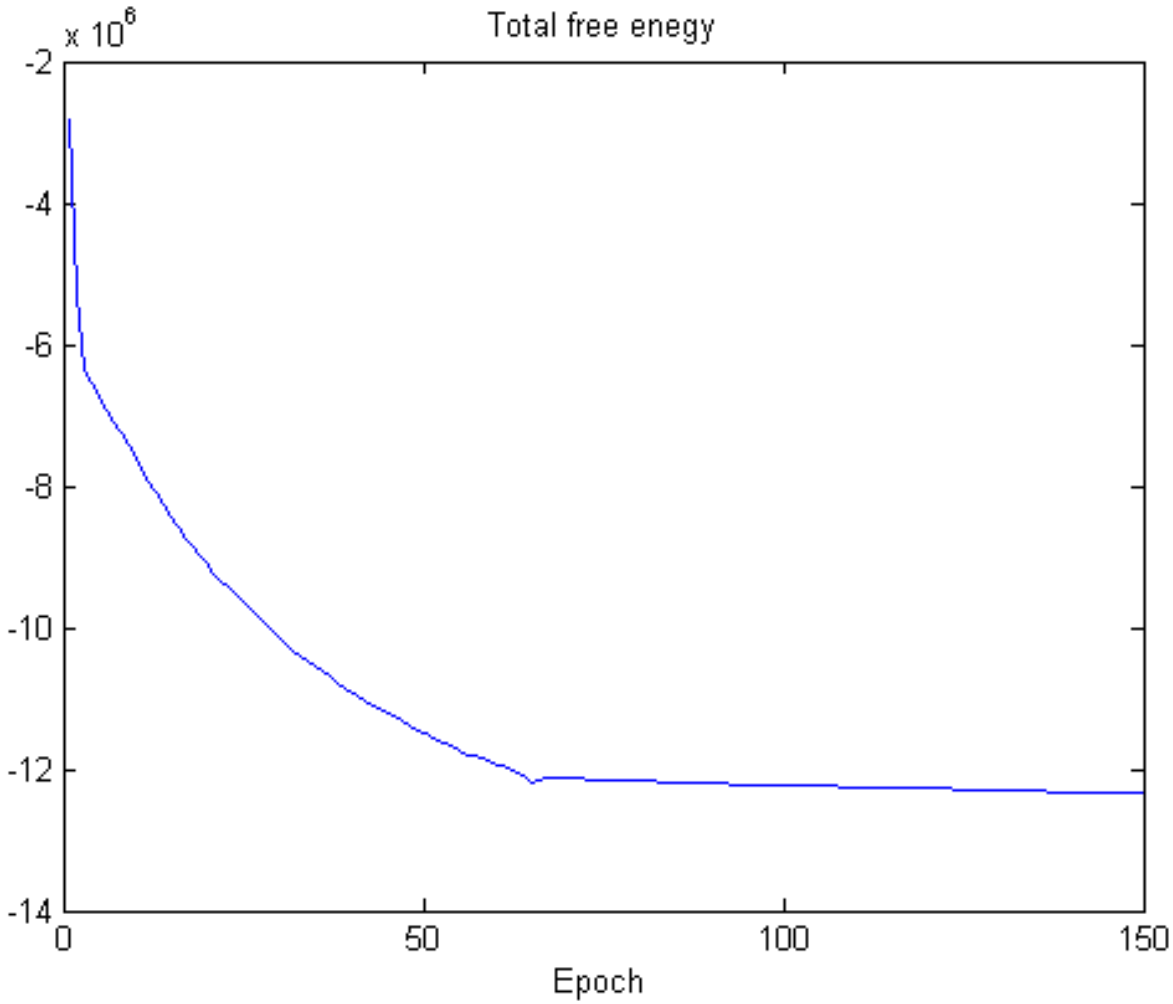
In the learning process we have recorded and plotted three valuation measurements:

- the reconstruction error,
- the total free energy,
- the classification error.

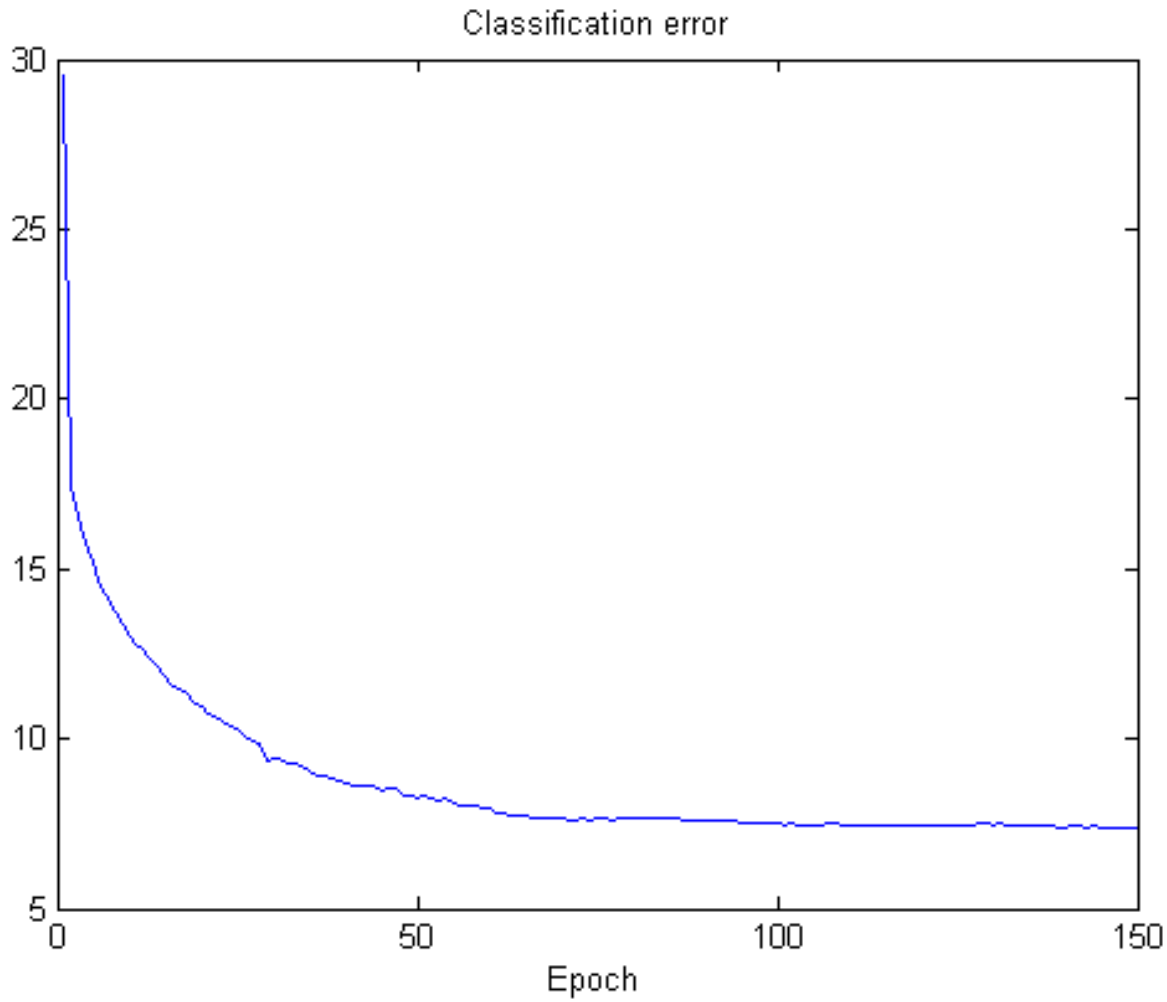
Reconstruction error during learning in generative training



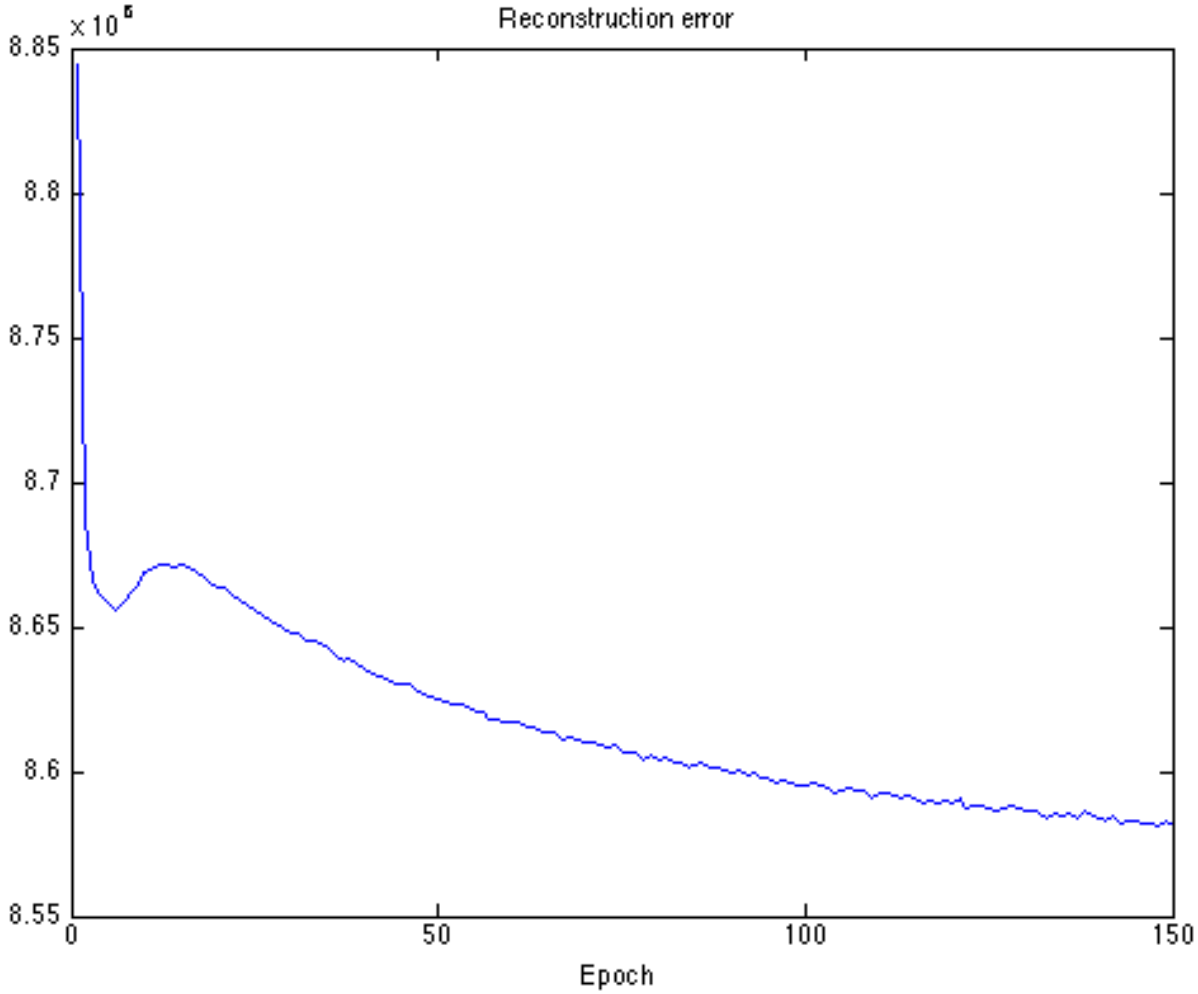
Free energy during learning in generative training



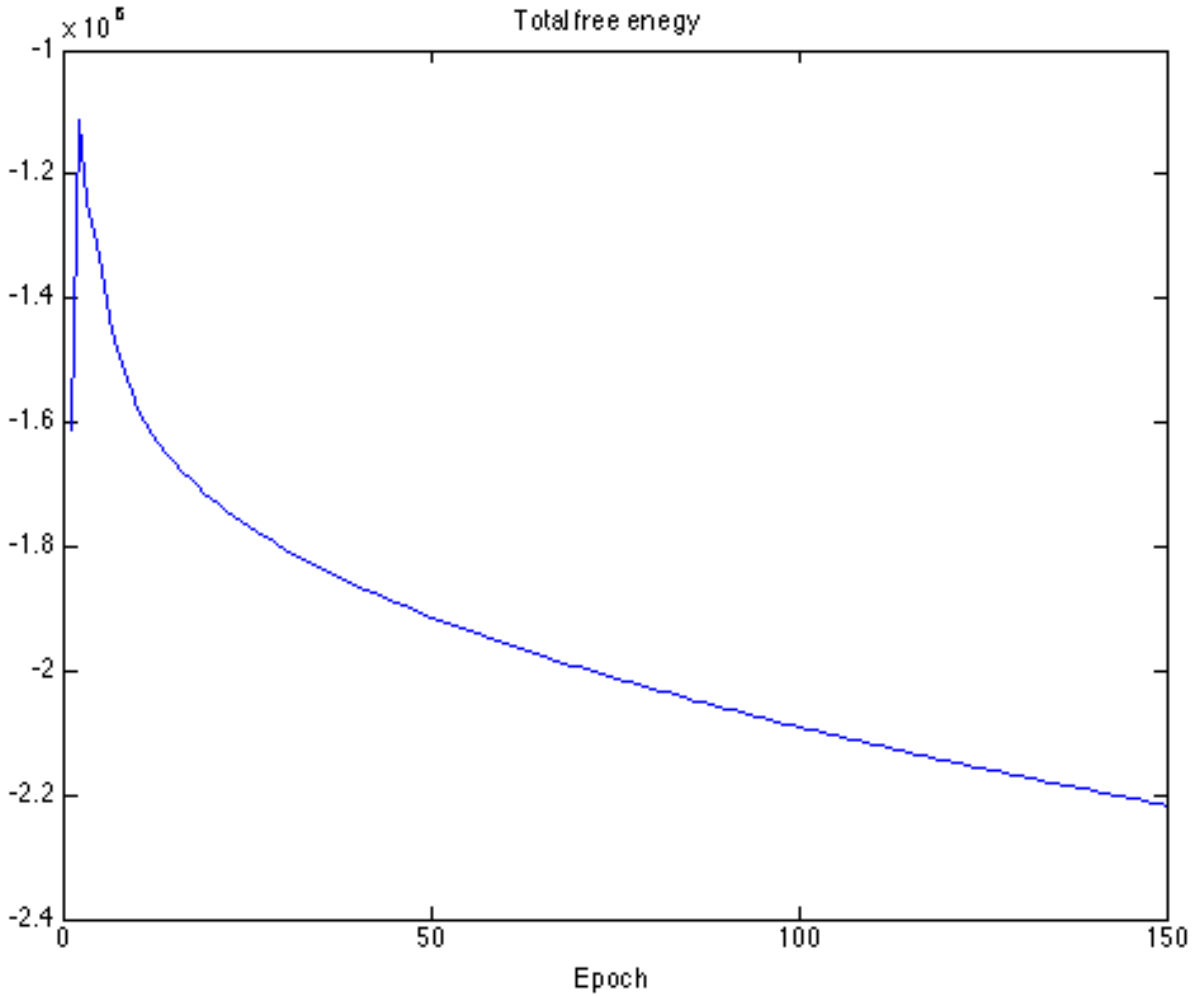
Classification error during learning in generative training



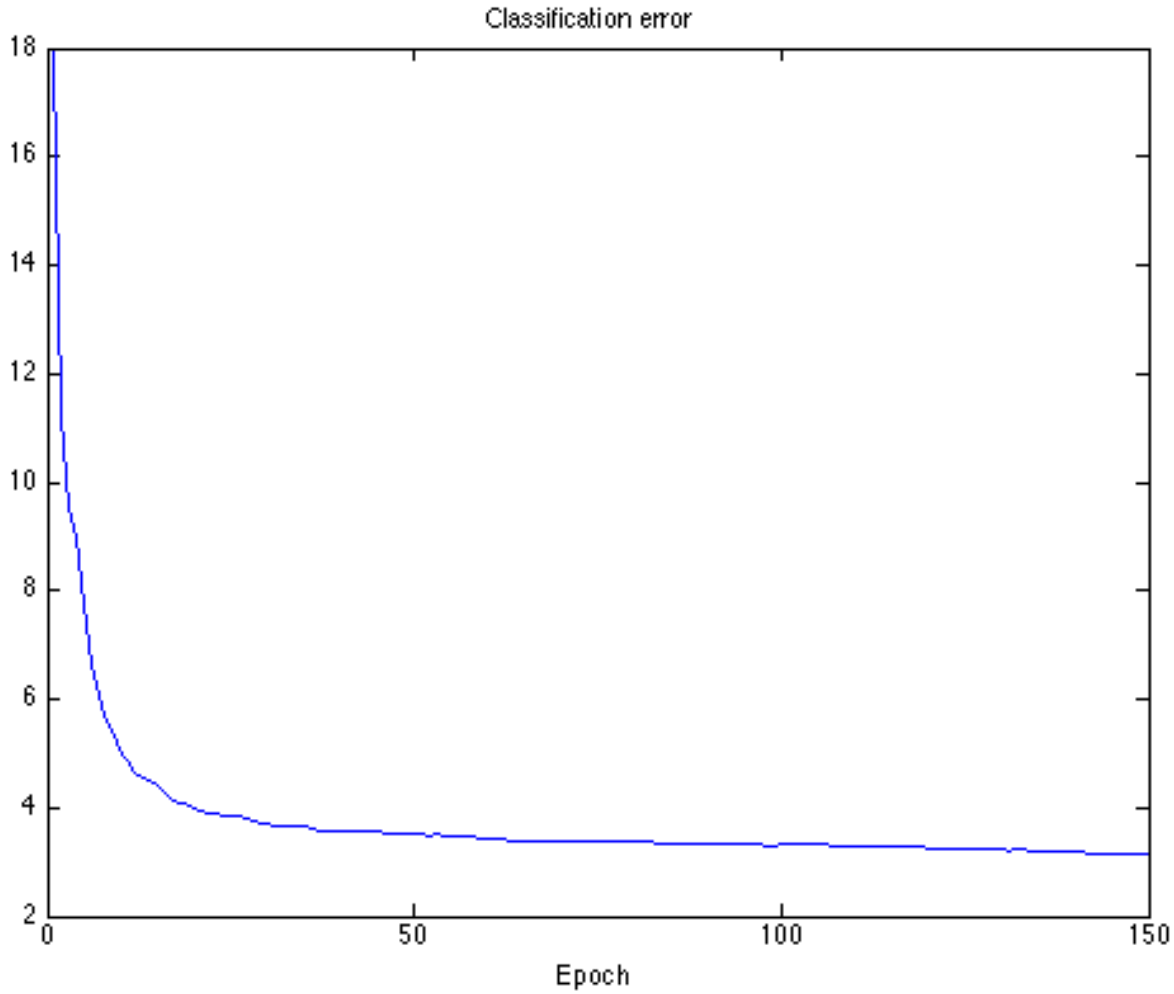
Reconstruction error during learning in discriminative training



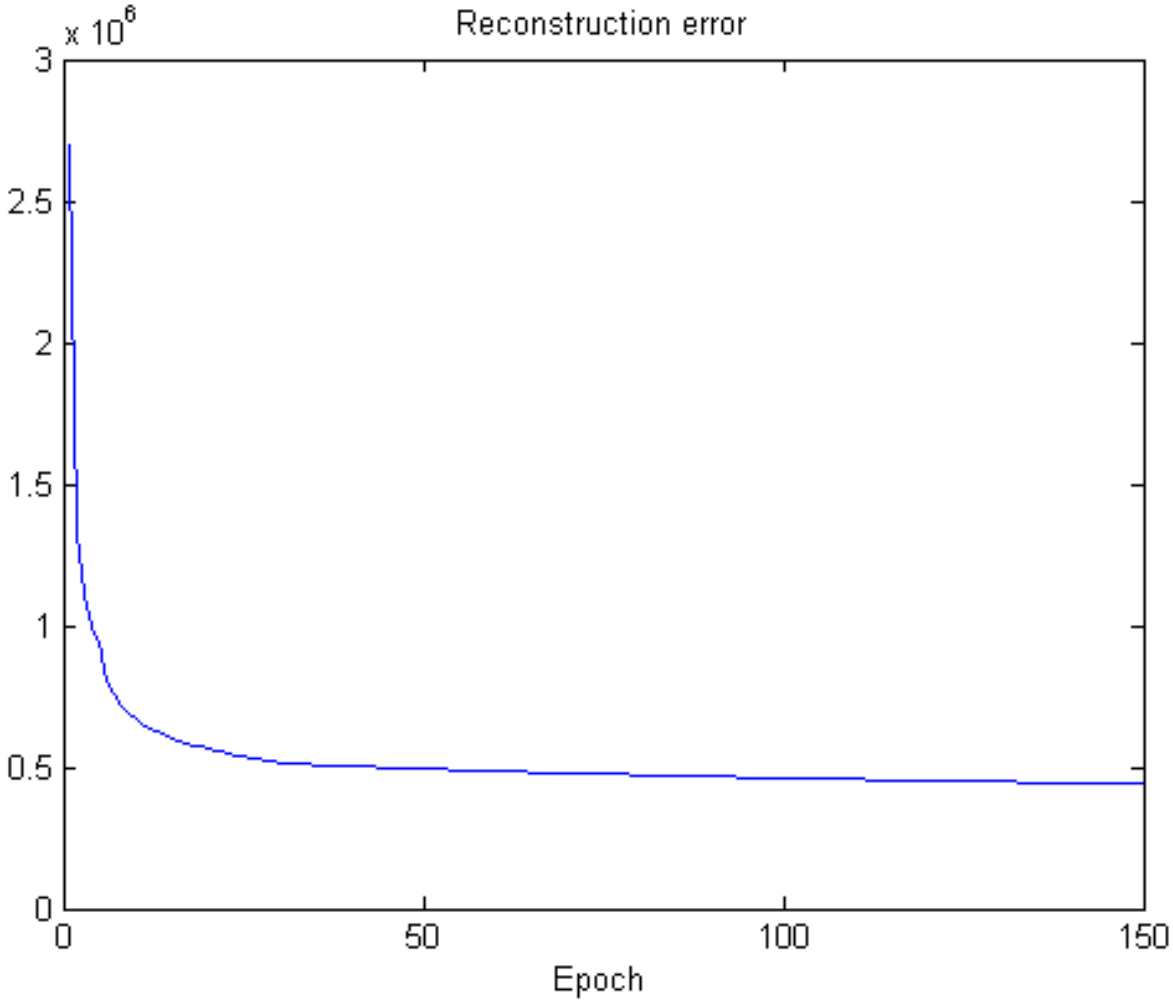
Free energy during learning in discriminative training



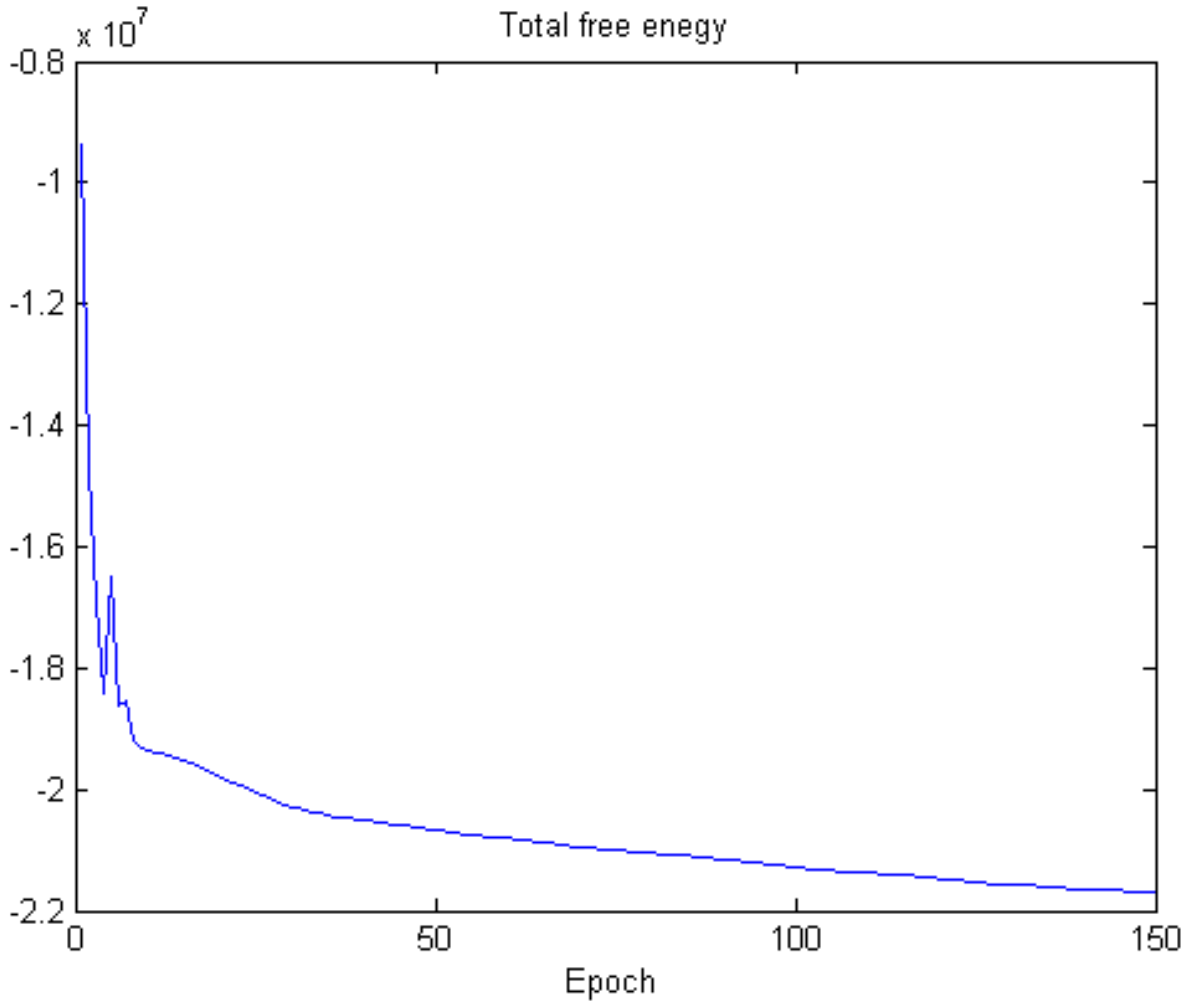
Classification error during learning in discriminative training



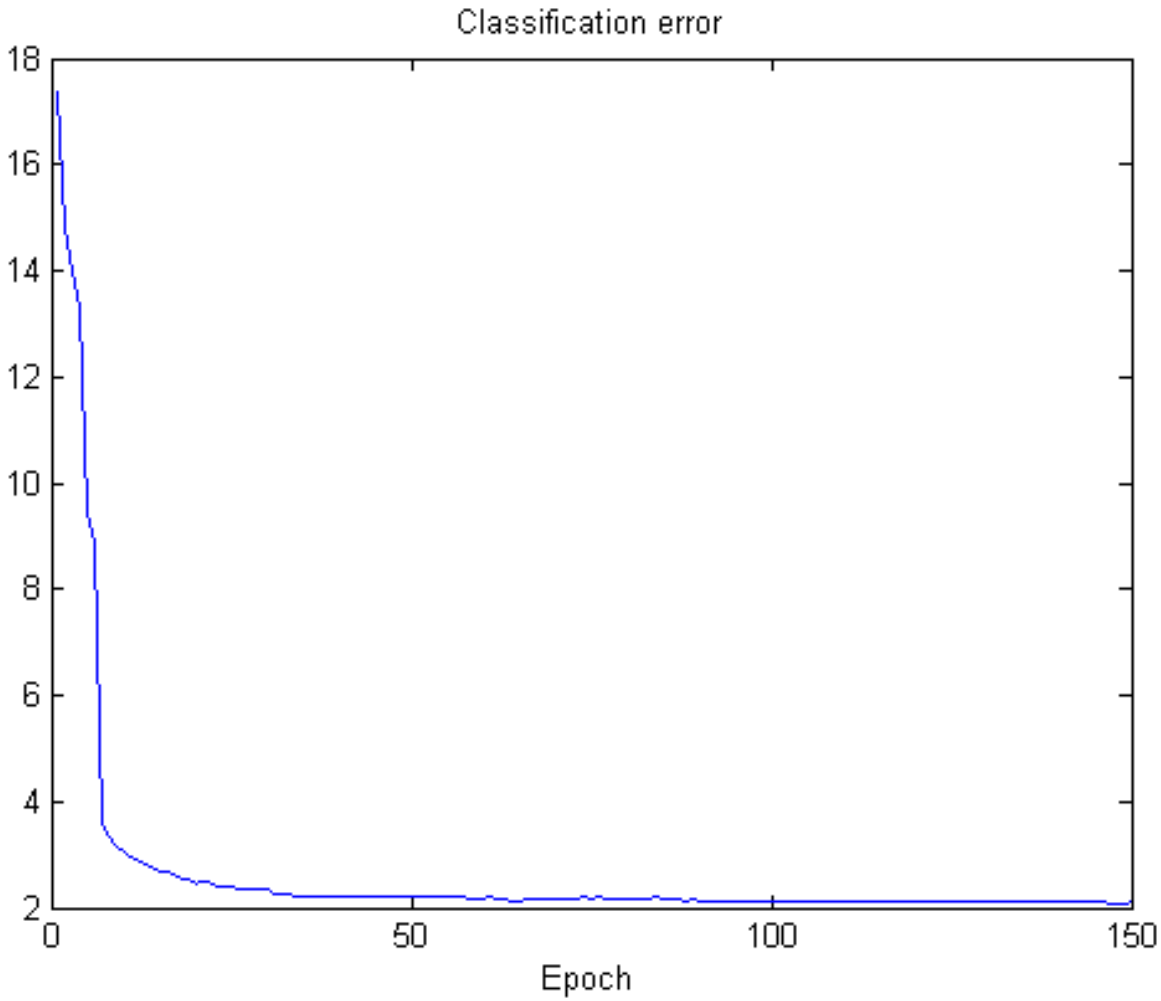
Reconstruction error during learning in hybrid training



Free energy during learning in hybrid training



Classification error during learning in hybrid training



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.
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.