Deep Belief Networks

Ashraf GHIYE

April 04, 2021

Contents

1 Introduction

The objective of this project is to study deep neural networks for the classification of handwritten numbers. We will compare the performances, in terms of rate of good classifications, of a pre-trained network (using Deep Belief Networks) and of a randomly initialized network, as a function of the number of training data, the number of layers of the network and finally the number the number of neurons per layer.

2 Data

2.1 AlphaDigits

This database from NYU[1], contains binary digits of size 20x16 that represent character "0" through "9" and capital "A" through "Z".

There are 39 examples of each class, thus the data frame will contain 1,404 rows each representing an image (39 examples of 36 classes) and 320 variables taking binary values 1 or 0 (black =1 and white = 0 for each pixel value of a 20x16 alpha-numeric image).

Below, we plot some examples to get a closer look into the dataset.

Figure 1: AlphaDigits examples

[1] http://www.cs.nyu.edu/ roweis/data.html

2.2 MNIST

The MNIST[2] database (Modified National Institute of Standards and Technology database) is a large collection of handwritten digits. It has a training set of 60,000 examples, and a test set of 10,000 examples. The digits have been size-normalized and centered in a fixed-size image. They were also centered in a 28x28 image by computing the center of mass of the pixels, and translating the image so as to position this point at the center of the 28x28 field.

In our case, we will binarize the images in order to get 0/1 pixels representing the background (in black) and the digit (in white).

Also, we will transform the labels into a one hot encoded vector, i.e. vector of size 10 for each image with all zeros except one at the index of digit.

Below, we show some examples from the dataset after binarization.

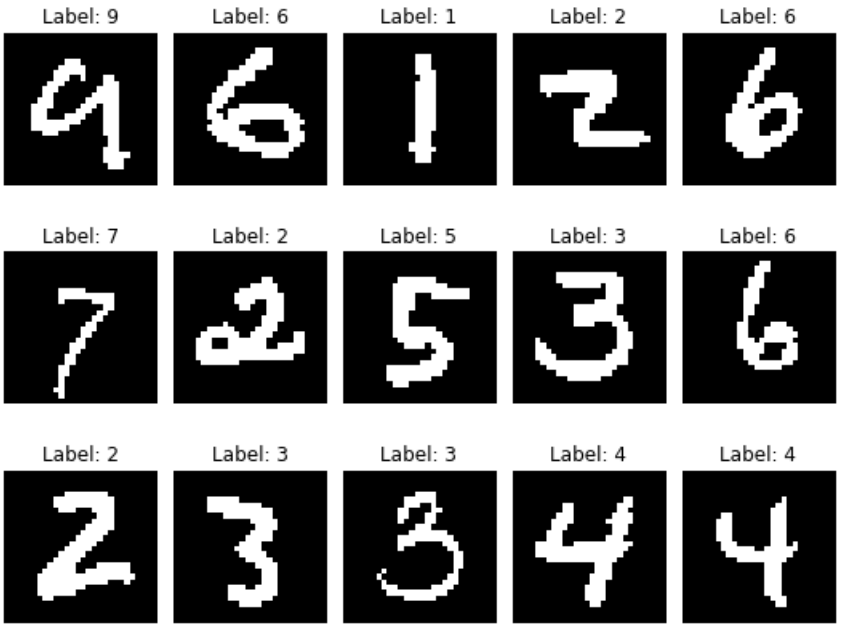


Figure 2: MNIST examples

[2] http://yann.lecun.com/exdb

3 Preliminary Work

In this part, we will use RBM and DBN as generative networks. For that, we will train these models using AlphaDigits in an unsupervised fashion. We will then assess the quality of generated characters visually and compare their quality as a function of the number of training examples, the dimension of hidden variables and the number of Gibbs sampling iterations.

3.1 Restricted Boltzmann Machine

Restricted Boltzmann Machine is a stochastic shallow network with p input variables v_i and q output (or latent) variables h_j.

We will only focus on the case of binary variables.

$$v_i \in \{0,1\} \quad 1 \leq i \leq p$$

$$h_j \in \{0,1\} \quad 1 \leq j \leq q$$

RBM models the joint probability distribution between those variables as a function of a potential energy $E(v,h)$:

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

Where $Z = \sum_{v,h} e^{-E(v,h)}$ is a normalization constant.

The potential energy can be written as a polynomial function of v and h:

$$E(v,h) = -\sum_{i=1}^{p} a_i v_i - \sum_{j=1}^{q} b_j h_j - \sum_{i,j} w_{ij} a_i b_j$$

Hence, the RBM is a parametric model characterized with $\Theta = (a_i, b_j, w_{ij})$ and q.

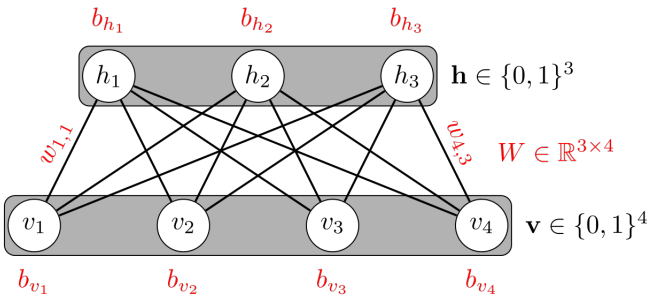


Figure 3: Restricted Boltzmann Machine with $p = 4$ and $q = 3$

Using Bayes formula, one can show easily that the conditional probability for a given output neuron $p(h_j|v)$ can be written as a simple sigmoid function:

$$p(h_j = 1|V) = \frac{1}{1 + e^{-(b_j + \sum_i w_{ij} v_i)}}$$

Conditioning on the input variables v, the latent variables h_j are independents from each other.

Boltzmann machines has shown to be universal approximators for probability distributions on **binary vectors**.

RBM is a parametric statistical model, meaning that to learn the parameters Θ, we will need some statistical estimators such as Maximum Likelihood Estimation. It is for this reason they were widely used when backpropagation was infeasible.

However, the distribution is only known up to an unknown constant Z. Its calculation is intractable due to combinatorial complexity.

Therefore, training such models will be done using the Contrastive-Divergence 1 algorithm. G. Hinton proposed this algorithm in 2002 to avoid the difficulty in computing the log-likelihood gradient. It uses MCMC methods to sample from the unknown distribution an estimator to be used in the gradient descent step.

3.1.1 RBM Analysis

After we train the RBM successfully on the AlphaDigits, it can provide a closed-form representation of the distribution underlying the training data. It can be used then as a generative model that allows sampling from the learned distribution (Fig.4).

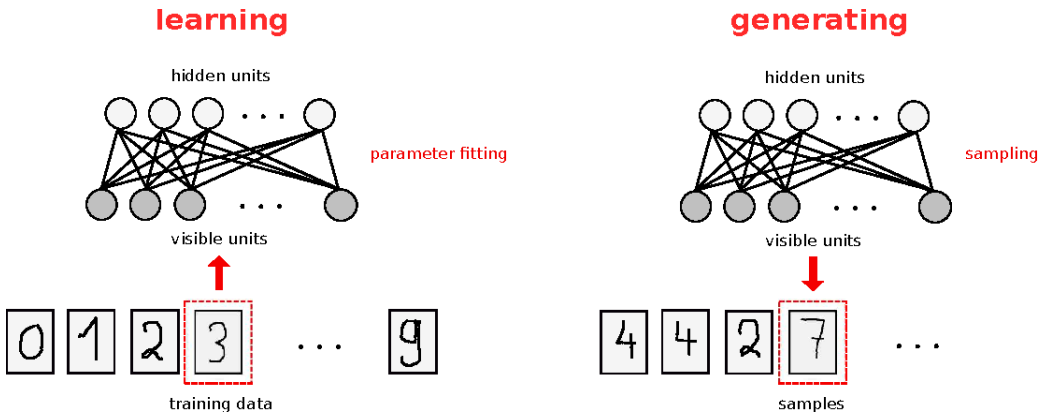
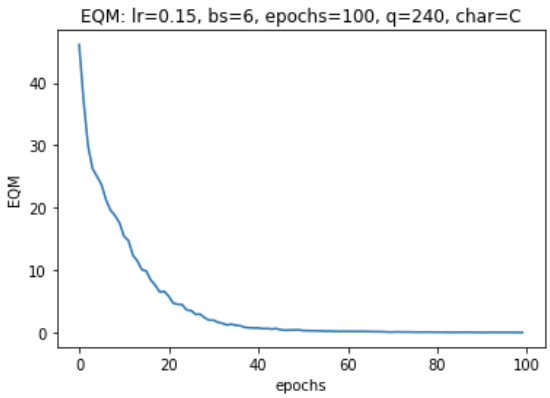


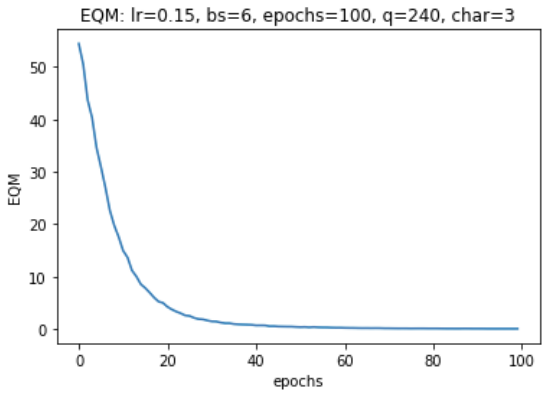
Figure 4: RBM for character generation

To test the convergence of the learning algorithm, we compute the reconstruction error after each epoch which consist of comparing all the original samples X with reconstructed versions X' with respect to the ecludian distance, EQM = $||X - X'||^2$.

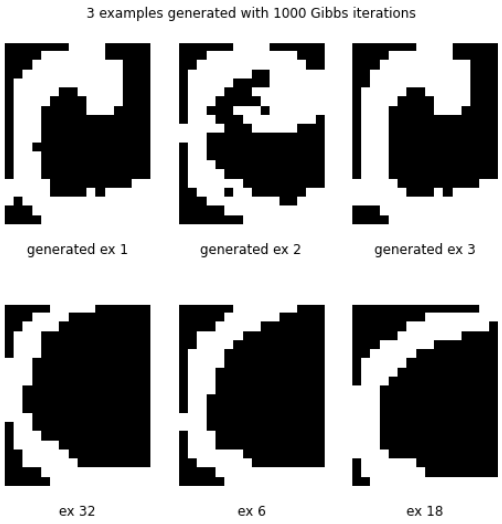
Here are the learning curves after training with the whole examples for two characters, i.e. we used the 39 examples for each of the character C and digit 3 to train two RBMs using CD-1:



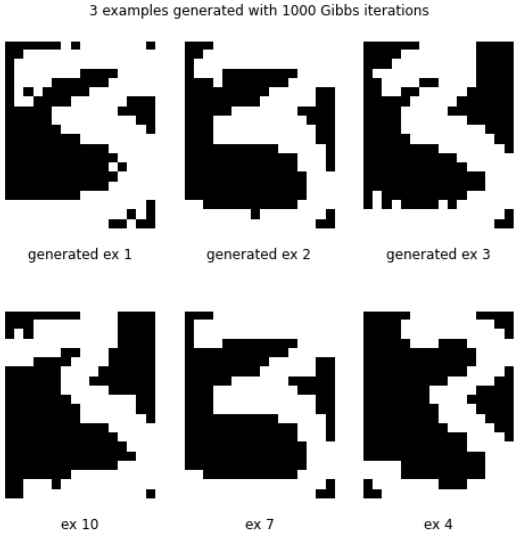
(a) Learning Curve 'C'



(b) learning Curve '3'



(c) Generating 'C' examples



(d) Generating '3' examples

Figure 5: Experimental Results

From the Fig.5, it is clear that the network was capable of regenerating good examples, meaning that we were able to learn the distribution of the input space (AlphaDigits). After learning the join distribution of input and output spaces using the RBM we were able to re-generate characters and digits through Gibbs sampling.

Note that, as for any learning algorithm, the hyper-parameters play a key role in the shape and speed of the learning curve. The optimal choice of these parameters will requires a grid-search to obtain the best configuration that minimizes the reconstruction loss and yields the best results.

However, we have run a sequential search of the best parameters, i.e. by fixing other parameters and selecting the best parameter one at a time. We started from a convenient configuration and ended up with the parameters shown in Fig.5 (a) and (b).

Here is the studied effects of the various hyper-parameters (for the learning algorithm and for the model itself):

- **Learning rate**: The magnitude of updates controls the speed of convergence. If the learning rate is set too low, training will progress very slowly towards the optimal solution. Nevertheless, too large steps can cause the model to converge too quickly to a suboptimal solution or worth: to diverge. Thus, this parameter requires a trade-off between the rate of convergence and overshooting.

- **Batch size**: It controls the accuracy of the estimate of the error gradient when training with a stochastic gradient descent. It is a trade-off between the speed of computation and the quality of estimation (of the gradient).

- **epochs**: The number of effective pass over the whole dataset, e.g. for a small learning rate, we need more epochs to have more time to learn more. The choice of this parameter depends on the batch size and learning rate. Also, on the dynamics in the learning curve.

- **q**: The dimension of latent space. Lower values will make the learning process more complex, while large values will make learning easier. But this will eliminate the benefits of dimensionality reduction. If we have an input dimension p of, say 320, then, would like to summarize the information in a much smaller dimension. Therefore we want q, the output dimension, to be equal to 240. Only then we will be able to compress with a factor of 1.5 the original information.

- **Gibbs iterations**: The number of sampling iterations to generate an image, the bigger this number is the better quality of reconstructed image we get. It is also a trade-off between the quality and the efficiency of the algorithm.

3.2 Deep Belief Network

The basic idea underlying these deep architectures is that the hidden neurons of a trained RBM represent relevant features of the observations, and that these features can serve as input for another RBM, see Fig.6 for an illustration. By stacking RBMs in this way, one can learn features from features in the hope of arriving at a high-level representation.

Thus a Deep Belief Network (DBN) is a multi-layer generative graphical model. DBNs have bi-directional connections (RBM-type connections) on the top layer while the bottom layers only have top-down connections. Mathematically put, a DBN with L layers has p_0 input units and $p_1 ... p_L$ units for each of the following layers.

$$v_i = h_i^{(0)} \in \{0,1\}^{p_0}$$

$$h_i^{(1)} \in \{0,1\}^{p_1}; \quad h_i^{(k)} \in \{0,1\}^{p_k}; \quad h_i^{(L)} \in \{0,1\}^{p_L}$$

Note that only the last layer can be considered as RBM:

$$p(h^{L-1}, h^L) = \frac{1}{Z} e^{-E(h^{L-1}, h^L)}$$

and the relation between the other layers are given by:

$$p(h^{i-1}, h^i) = \prod_{j=1}^{p_i} p(h_j^{i-1}, h^i)$$

and $p(h_j^{i-1} = 1 | h^i) = \text{sigmoid}(b_j^{i-1} + \sum_k w_{jk}^i h_k^i)$.

On a side note, a Deep Belief Network is not a Deep Boltzmann Machine. In which the latter is a concatenation of undirectional RBM networks.

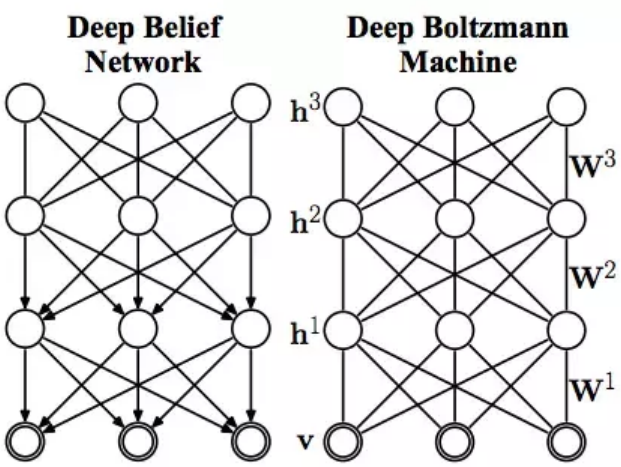


Figure 6: Comparison between DBN and DBM

A DBN can be trained in a greedy unsupervised way, by training separately each RBM from it, in a bottom to top fashion, and using the hidden layer as an input layer for the next RBM.

This greedy approach consider a DBN as a stack of RBM, we then iterate over each layer and run a CD-1 algorithm to update its parameters a, b and W (Fig.7).

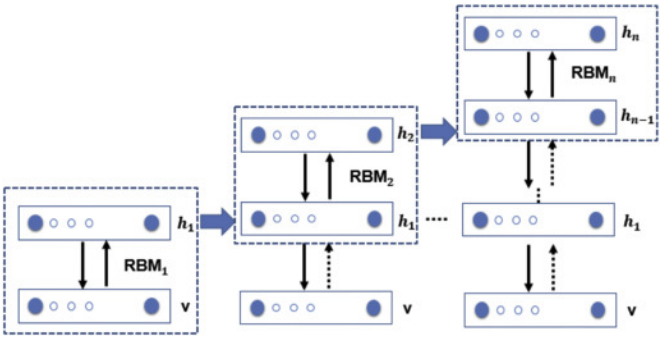


Figure 7: Training DBN

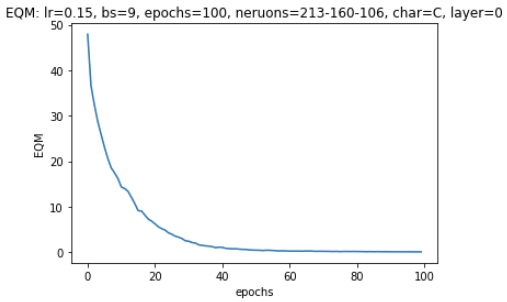
The advantage of a DBN over RBM is that via deep architecture, we are able to approximate complex joint distributions in an efficient manner comparing to a simple RBM. However, it comes with a computational cost (more parameters to estimate) and thus it needs more data to be learnt efficiently.

3.2.1 DBN Analysis

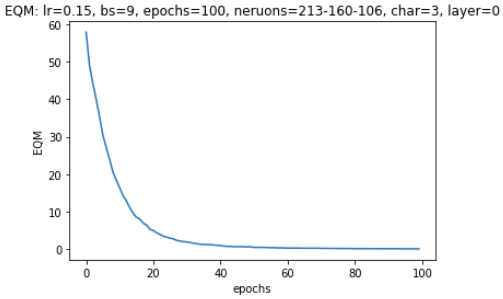
Figure 8 shows the experimental results of learning a DBN on AlphaDigits and using the learned model to generate some examples.

As in the case of an RBM, we used heuristics in the choice of hyper-parameters by keeping the best learning parameters from the previous experiments and choosing a three-layer DBN with $\frac{2}{3}p$, $\frac{1}{2}p$ and $\frac{1}{3}p$ neurons respectively.

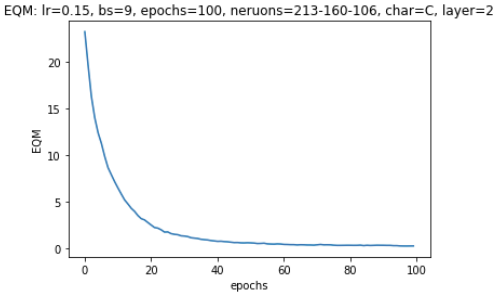
By comparing the quality of generated examples between a DBN and an RBM, we can see that the DBN has generated more realistic digits and characters. The RBM also gave good results, yet with more noise. Moreover, the DBN has learnt a lower-dimensional representation ($\frac{p}{3}$) of the input space (which can be regarded as higher-abstract features) compared to the RBM ($\frac{p}{1.5}$).



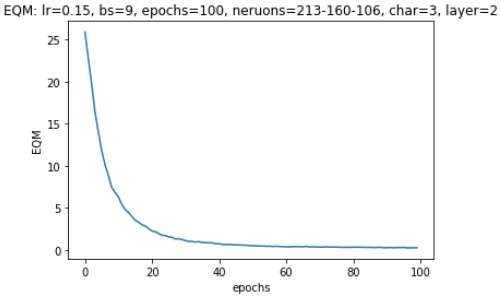
(a) Learning Curve layer 0



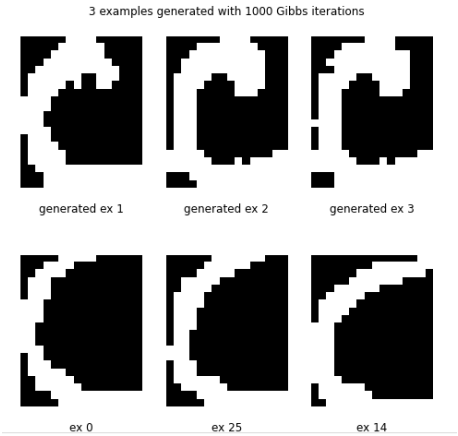
(b) Learning Curve layer 0



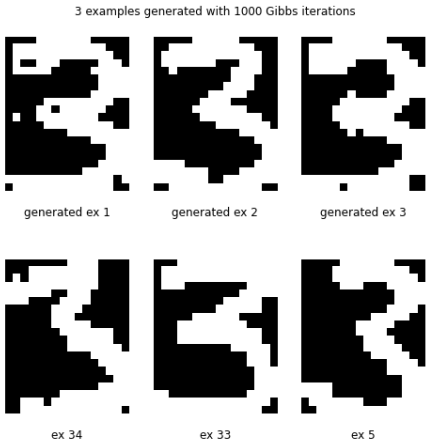
(c) Learning Curve layer 2



(d) Learning Curve layer 2



(e) Generating 'C' examples



(f) Generating '3' examples

Figure 8: Experimental Results

4 Primary Work

It is an important property that single as well as stacked RBMs can be reinterpreted as deterministic feed-forward neural networks. When viewed as neural networks they are used as functions mapping the observations to the expectations of the latent variables in the top layer. These can be interpreted as the learned features, which can, for example, serve as inputs for a supervised learning system. Furthermore, the neural network corresponding to a trained RBM or DBN can be augmented by an output layer where the additional units represent labels (e.g., corresponding to classes) of the observations. Then we have a standard neural network for classification that can be further trained by the standard supervised learning algorithms (backpropagation). It has been argued that this initialization (or unsupervised pre-training) of the feed-forward neural network weights based on a generative model helps to overcome some of the problems that have been observed when training multi-layer neural networks [25].

We will experiment this results ourselves using the MNIST dataset, thus we will consider two neural networks, one is pre-trained as a DBN and the other is randomly initialized. We will then compare both models in term of accuracy.

4.1 Deep Neural Network

4.2 Results

5 Conclusion