DeepLearnLab Tutorial and API

Malcolm Greaves, Dan Howarth, Ziyi Zhu

May 1st, 2013

Contents

List of Figures

List of Tables

Chapter 1

Introduction

An open source implementation of deep belief networks is available at
`www.github.com/howarth/DeepLearnLab`

Deep belief networks (DBN) are powerful generative models for learning a representation of data. In this document we will explain three main topics: (1) what DBNs are and how they work, (2) how to learn a DBN with DeepLearnLab and (3) how to run a DBN demonstration on the MNIST handwritten character dataset.

Chapter 2

Deep Belief Networks

Restricted boltzmann machines (RBM) are the fundamental building block of deep belief networks. The first section in this chapter explains what RBMs are. The second section explains how to stack RBMs together in order to create deep belief networks.

2.1 Restricted Boltzmann Machines

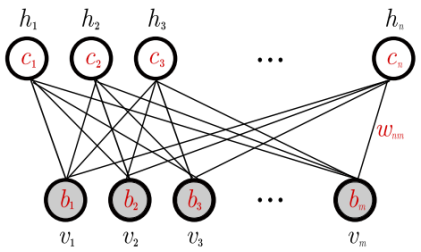
2.1.1 Representation

A restricted boltzmann machine consists of two layers of nodes with connections between layers. One layer is the visible layer, or in other words the layer which is the data. The second layer is the hidden layer which is composed of latent variables. Between the two layers there are weighted connections. Each hidden node is connected to every visible node. There are no connections between visible nodes nor is are there any connections between hidden nodes. This is what sets a restricted boltzmann machine apart from a boltzmann machine. This is visually depicted in figure 2.1.

The values of the hidden and visible nodes can be either 0 or 1. The value of a weight from v_i to h_j can be any number and corresponds to how much the RBM believes both h_j and v_i should take the value 1 together. This can be seen through the definition of the probability of a configuration of the model.

The energy of the model is defined as in equation (2.1)

Figure 2.1: A restricted boltzmann machine



$$E(v,h) = -\sum_i b_i v_i - \sum_j c_j h_j - \sum_{i,j} v_i w_{ij} h_j \tag{2.1}$$

Where b_i is the bias of the ith visible unit and c_j is the bias of the jth hidden unit. Given the energy equation we can define the probability of a setting to the RBM. This definition of probability is shown in equation (2.2).

$$P(v,h) = \frac{e^{-E(v,h)}}{\sum_{v',h'} e^{-E(v',h')}} \tag{2.2}$$

2.1.2 Learning Process

An efficient way to learn RBMs is with contrastive divergence. Contrastive divergence learns by iteratively changing the weights so that units that should be on together become more tightly connected. The weight update for contrastive divergence is shown in equation (2.3)

$$\delta w_{i,j} = E[v_i^0 h_j^0] - E[v_i^1 h_j^1] \tag{2.3}$$

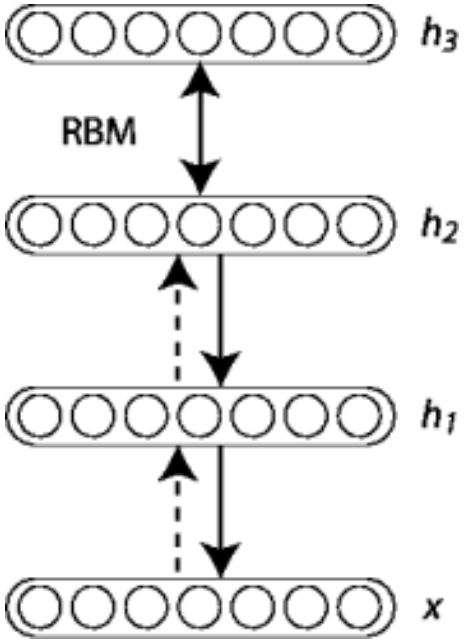
2.2 Stacking RBMs to create DBNs

A DBN is created by "stacking" RBMs on top of each other. In other words you build an RBM based on the data. Then you take that RBM's output and use it as input for the next level RBM. This is done as many times as you'd like until you are satisfied with your model. A visual depiction of this is show in figure 2.2

2.3 Further reading

See our reading list document in the git repository at `doc/supplementary_reading_list.pdf`

Figure 2.2: Stacking RBMs to create a DBN



(http://deeplearning.net/tutorial/DBN.html: Accesses April 28, 2013. Copyright 2008-2010, LISA lab)

Chapter 3

API

3.1 Training an DBN

In order to train an RBM you use `createRBM` to create the RBM and then `trainRBM` to train the RBM. These functions are defined below.

To create an DBN just stack the RBMs.

3.2 Functions

3.2.1 createRBM

```
[rbm] =  createRBM( n_v, n_h, t_v, scale, sparsity, sparsity_decay)
creates a definition of an RBM model based on the given parameters

INPUTS:
  n_v.............: number of vissible units
  n_h.............: number of hidden units
  t_v.............: type of visible units
                     possible types:
                     'binary'
                     'gauss'
  scale...........: scale of initial weights
  sparsity........: desired sparsity level of hidden units
  sparsity_decay..: sparsity decay value, in range [0, 1]

OUTPUTS:
  rbm.....: an instatiation of a rbm model given the inputs
```

3.2.2 trainRBM

```
[rbm ] = trainRBM( rbm, data, batch_size, nepochs, learn_rate)
```

```
trains an rbm based on the given parameters

  INPUTS:
    rbm.........: an rbm created by createRBM
    data........: (n x p) data matrix
    batch_size..: number of examples in each batch
    nepochs.....: number of times to go through the data
    learn_rate..: the learning rate of the training.

  OUTPUTS:
    rbm.........: the trained rbm
```

Chapter 4

MNIST handwritten character demo

Download the MNIST dataset at `http://www.cs.nyu.edu/ roweis/data/mnist_all.mat`. Then run `rbm = trainRBMMNIST(location)` with the location of the data as input.