presentation-dnn

Juan Carlos Cuevas-Tello
Universidad Autónoma de San Luis Potosí

Manuel Valenzuela-Rendón
Tecnológico de Monterrey

Deep Neural Networks

Dr. Juan Carlos Cuevas-Tello, Manuel Valenzuela-Rendon,
Juan A. Nolazco-Flores

Link paper: http://arxiv.org/abs/1603.07249
DOI: 10.13140/RG.2.1.2420.6481
March 2016

Introduction to DNNs

- The core of Deep Neural Networks (DNNs) are the Restricted Boltzmann Machines (RBMs) proposed by Smolensky et al. (1986).

Introduction to DNNs

- The core of Deep Neural Networks (DNNs) are the Restricted Boltzmann Machines (RBMs) proposed by Smolensky et al. (1986).
- Widely studied by Hinton et al. (2006-2010).

Introduction to DNNs

- The core of Deep Neural Networks (DNNs) are the Restricted Boltzmann Machines (RBMs) proposed by Smolensky et al. (1986).
- Widely studied by Hinton et al. (2006-2010).
- The term *deep* comes from Deep Beliefs Networks (DBN) – (Hinton, 2006).

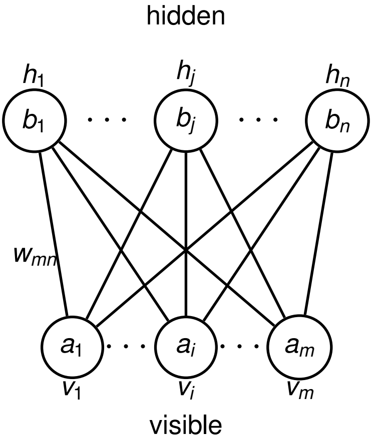
Introduction to DNNs

- The core of Deep Neural Networks (DNNs) are the Restricted Boltzmann Machines (RBMs) proposed by Smolensky et al. (1986).
- Widely studied by Hinton et al. (2006-2010).
- The term *deep* comes from Deep Beliefs Networks (DBN) – (Hinton, 2006).
- The term *Deep Learning* (DL) is becoming popular in the machine learning literature.

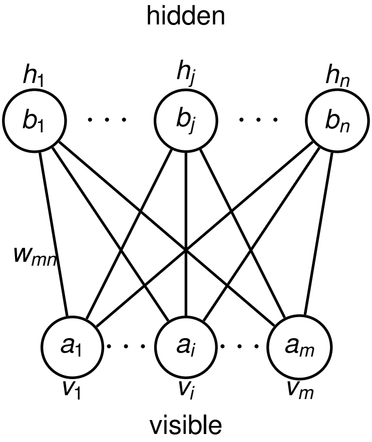
Introduction to DNNs

- The core of Deep Neural Networks (DNNs) are the Restricted Boltzmann Machines (RBMs) proposed by Smolensky et al. (1986).
- Widely studied by Hinton et al. (2006-2010).
- The term *deep* comes from Deep Beliefs Networks (DBN) – (Hinton, 2006).
- The term *Deep Learning* (DL) is becoming popular in the machine learning literature.
- DL is mainly based on RBMs and DBN.

RBMs



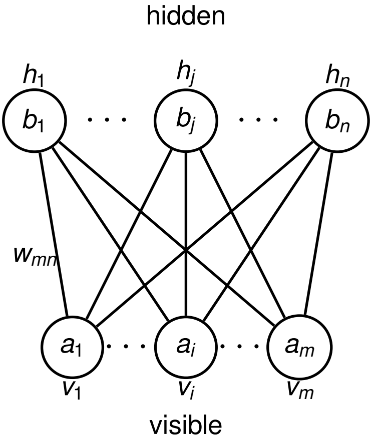
RBMs



$$p(h_j = 1|\mathbf{v}) = \mathbf{sig}\left(b_j + \sum_i v_i w_{ij}\right) \tag{1}$$

$\langle v_i h_j \rangle^0$

RBMs



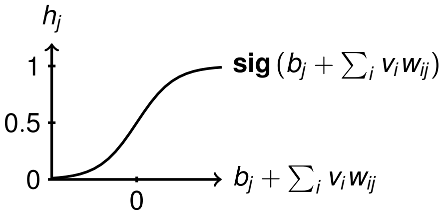
$$p(h_j = 1|\mathbf{v}) = \mathbf{sig}\left(b_j + \sum_i v_i w_{ij}\right) \tag{1}$$

$$\langle v_i h_j \rangle^0$$

$$p(v_i = 1|\mathbf{h}) = \mathbf{sig}\left(a_i + \sum_j h_j w_{i,j}\right) \tag{2}$$

$$\langle h_j v_i \rangle^0$$

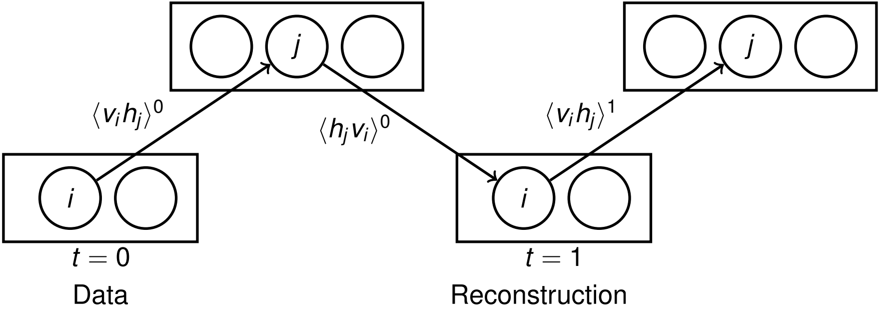
Building Block of RBMs



- The building block of a RBM is a binary stochastic neuron.
- The data to be learned is set at the visible layer; this data can be an image, a signal, etcetera.
- The state of the neurons at the hidden layer is obtained by $p(h_j = 1|\mathbf{v}) = \mathbf{sig}\left(b_j + \sum_i v_i w_{ij}\right)$

Constrastive Divergence Algorithm (CD)

Unsupervised learning algorithm, where ε is the learning rate



$$\Delta w_{ij} = \varepsilon \left(\langle v_i h_j \rangle^0 - \langle v_i h_j \rangle^1 \right) ; \tag{3}$$

$$\Delta a_i = \varepsilon \left(v_i^0 - v_i^1 \right) ; \tag{4}$$

$$\Delta b_j = \varepsilon \left(h_j^0 - h_j^1 \right) ; \tag{5}$$

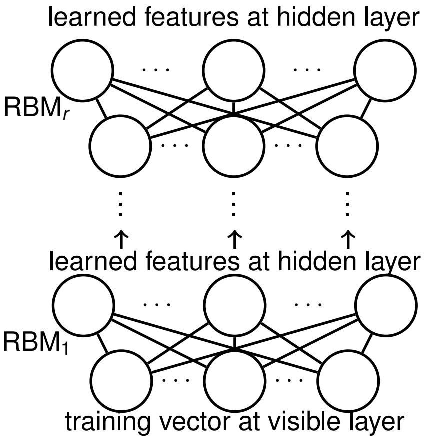
Pseudocode CD

ALGORITHM 1: Pseudocode of the Contrastive Divergence Algorithm, CD_k.

```
Set the visible units to a training vector ;
for k ← 1 to maximum of iterations do
    for s ← 1 to size of training data do
        Update all the hidden units in parallel with Eq. 1 ;
        Update all visible units in parallel to get "reconstructions" with Eq. 2 ;
        Update all hidden units again with Eq. 1 ;
        Update weights and biases with Eqs. 3–5 ;
        Select another training vector ;
    end
end
```

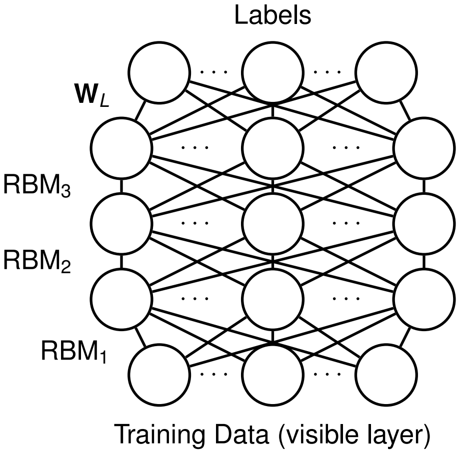
Deep Belief Networks (DBN)

Stacking RBMs



Hybrid DBN -> DNN

Supervised Learning



Toolbox for DNN

- A publicly available toolbox for MATLAB® developed by Tanaka et al. (2014)
- Download online.
- `/DeepNeuralNetwork/`
- `/DeepNeuralNetwork/mnist`

MNIST: benchmark

THE MNIST DATABASE of handwritten digits

- Yann LeCun, Courant Institute, NYU
- Corinna Cortes, Google Labs, New York
- Christopher J.C. Burges, Microsoft Research, Redmond

Link to website

DNNs & MNIST database

DNNs for handwritten digits recognition

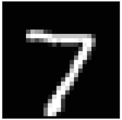
10 label units

$\mathbf{W}_L$

RBM_2 800 hidden units

RBM_1 800 hidden units

Visible units 28×28 pixels digit image

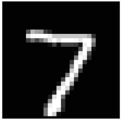


Tanaka et al. (2014)

DNNs & MNIST database

DNNs for handwritten digits recognition

	10 label units
$\mathbf{W}_L$	
RBM_2	800 hidden units
RBM_1	800 hidden units
Visible units	28×28 pixels digit image



- Train: 60,000 examples (patterns)
- Test: 10,000 examples
- 3 days computing time
- PC: 4 CPUs 2.3GHz with 16 cores; 384 GB memory

Tanaka et al. (2014)

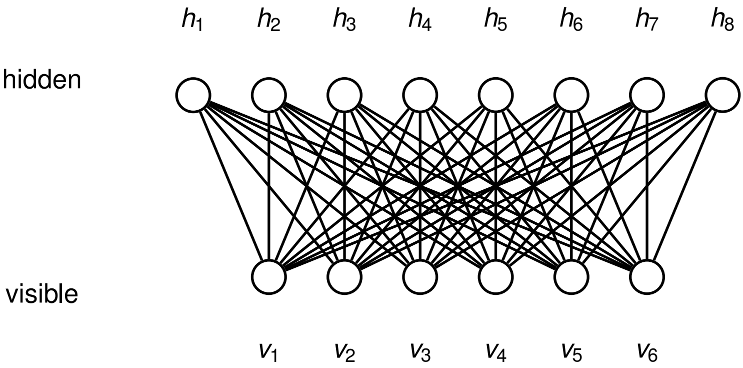
ErrorRate (training) = 0.196%; Tanaka et al. reported 0.158%

ErrorRate (testing) = 1.6%; Tanaka et al. reported 1.76%

MNIST – the best classifier

Classifier:	committee of 35 conv. net 1-20-P-40-P-150-10
Preprocessing:	width normalization
Test error rate (%):	0.23
Reference:	Ciresan et al. CVPR 2012

DNN Example 1: unsupervised learning




```
Pattern=[
       1   1   0   0   0   0 ;
       0   0   1   1   0   0 ;
       0   0   0   0   1   1 ];
Inputs = 6;     % #no variables as input
```

DNN Example 1: Matlab/Octave code

```
TestData = TrainData;
nodes = [Inputs 8]; % [#inputs #hidden]
bbdbn = randDBN( nodes, 'BBDBN' ); % Bernoulli-Bernoulli RBMs
nrbm = numel(bbdbn.rbm);
opts.MaxIter = 50; % meta-paramters or hyper-parameters
opts.BatchSize = 1;
opts.Verbose = false;
opts.StepRatio = 2.5;
opts.object = 'CrossEntorpy';
%Learning stage
opts.Layer = nrbm-1;
bbdbn = pretrainDBN(bbdbn, TrainData, opts);
%Testing stage
H = v2h( bbdbn, TestData); % visible layer to hidden layer
out = h2v(bbdbn,H); % hidden to visible layers (reconstruction)
```

DNN Example 1: Output

```
TrainData =
                  1   1   0   0   0   0
                  0   0   1   1   0   0
                  0   0   0   0   1   1
out =
 9.9e-01 9.9e-01 1.0e-04 1.2e-04 2.5e-05 2.6e-05
 5.0e-04 4.3e-04 9.9e-01 9.9e-01 6.1e-04 6.0e-04
 4.2e-06 3.6e-06 5.9e-06 5.4e-06 9.9e-01 9.9e-01
round(out) =
            1   1   0   0   0   0
            0   0   1   1   0   0
            0   0   0   0   1   1
```

DNN Example 2: Predicting

Supervised learning

2 label units

$\mathbf{W}_L$

RBM_1

20 hidden units

6 visible units

Visible units

```
Pattern1 = [
             0  0  0  0  0  1  0  1;
             0  0  0  0  1  0  1  0;
             0  0  0  1  0  1  0  0;
             0  0  1  0  1  0  0  0;
             0  1  0  1  0  0  0  0;
             1  0  1  0  0  0  0  0];
```

DNN Example 2: Matlab/Octave code

```
Inputs = 6; % #no variables as input
Outputs = 2; % #no. variables as ouputs
TrainData = Pattern(:,1:Inputs); % get the training data, inputs
TrainLabels = Pattern(:,Inputs+1:Inputs+Outputs); % show the labels
TestData = TrainData; % we test with the same training data
TestLabels = TrainLabels;
nodes = [Inputs 20 Outputs]; % [#inputs #hidden #outputs]
bbdbn = randDBN( nodes, 'BBDBN' ); % Bernoulli-Bernoulli RBMs
nrbm = numel(bbdbn.rbm);Âą
%Learning stage (hyperparametres as previous example)
opts.Layer = nrbm-1;
bbdbn = pretrainDBN(bbdbn, TrainData, opts);
bbdbn= SetLinearMapping(bbdbn, TrainData, TrainLabels);
opts.Layer = 0;
bbdbn = trainDBN(bbdbn, TrainData, TrainLabels, opts);
```

DNN Example 2: Output

```
Testing...
fprintf( 'Testing...\n' );
out = v2h( bbdbn, TestData );
[TestData out]
ans =
          0.0 0.0 0.0 0.0 0.0 1.0 0.0031 0.9881
          0.0 0.0 0.0 0.0 1.0 0.0 0.9911 0.0011
          0.0 0.0 0.0 1.0 0.0 1.0 0.0044 0.0112
          0.0 0.0 1.0 0.0 1.0 0.0 0.0092 0.0073
          0.0 1.0 0.0 1.0 0.0 0.0 0.0065 0.0002
          1.0 0.0 1.0 0.0 0.0 0.0 0.0003 0.0041
```

DNN Example 2: Output (cont...)

```
[TestData round(out)]
ans =
        0     0     0     0     0     1     0     1
        0     0     0     0     1     0     1     0
        0     0     0     1     0     1     0     0
        0     0     1     0     1     0     0     0
        0     1     0     1     0     0     0     0
        1     0     1     0     0     0     0     0
```

DNN Example 3

4 inputs, 2 outputs:

```
Training...
ans =
          0   0   0   0   0   1
          0   0   0   0   1   0
          0   0   0   1   0   0
          0   0   1   0   0   0
          0   1   0   0   0   0
          1   0   0   0   0   0
Testing...
ans =
          0.0  0.0  0.0  0.0  0.49922 0.49922
          0.0  0.0  0.0  0.0  0.49922 0.49922
          0.0  0.0  0.0  1.0  0.00993 0.00993
          0.0  0.0  1.0  0.0  0.01070 0.01070
          0.0  1.0  0.0  0.0  0.01142 0.01142
          1.0  0.0  0.0  0.0  0.01109 0.01109
```

DNN Example 4: XOR

XOR is a non-linear problem, supervised learning.

1 label unit

$\mathbf{W}_L$ $\updownarrow$

RBM_1 12 hidden units

$\updownarrow$

Visible units 2 visible units

x_1	x_2	$x_1 \oplus x_2$
0	0	0
0	1	1
1	0	1
1	1	0

DNN Example 4: XOR Output

```
    nodes = [2 3 3 1]; % [#inputs #hidden #hidden #outputs]
    pts.MaxIter = 10;
    opts.BatchSize = 4;
    opts.Verbose = true;
    opts.StepRatio = 0.1;
    opts.object = 'CrossEntropy';
    TestData = TrainData;

out' = 0.49994 0.49991 0.50009 0.50006

More iterations: opts.MaxIter = 100
out' = 0.47035 0.51976 0.49161 0.50654

More hidden neurons: nodes = [2 12 12 1]
out' = 0.118510 0.906046 0.878771 0.096262

Still more iterations: opts.MaxIter = 1000
out' = 0.014325 0.982409 0.990972 0.012630
Elapsed time: 32.607048 seconds
```

DNN Example 4: XOR Output (cont...)

```
Less hidden nodes: nodes = [2 12 1]; opts.MaxIter = 1000
out' = 0.043396 0.950205 0.947391 0.059305
Elapsed time: 23.640984 seconds

Less iterations: opts.MaxIter = 100
out' = 0.16363 0.80535 0.82647 0.20440
Elapsed time: 2.617439 seconds

The learning rate, opts.StepRatio = 0.01, opts.MaxIter = 1000
Similar results to the previous experiment

opts.StepRatio = 2.5, opts.MaxIter = 100
out' = 0.022711 0.955236 0.955606 0.065202
Elapsed time: 0.806343
```