CSC321 Tutorial 10: Review of Restricted Boltzman Machines and multiple layers of features (stacked RBMs). and Explanation of Assignment 4.

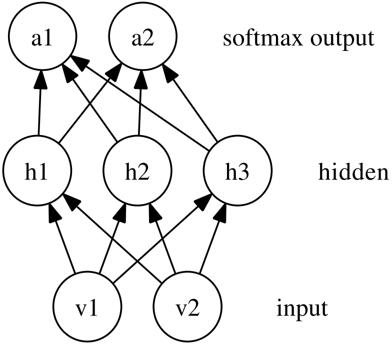
(Background slides based on Lecture 17-21)

Yue Li
Email: yueli@cs.toronto.edu

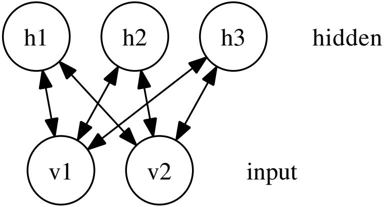
Wed 11-12 March 26
Fri 10-11 March 28

Oversimplified conceptual comparison b/w FFN and RBM

Feedforward Neural Network - **supervised** learning machine:



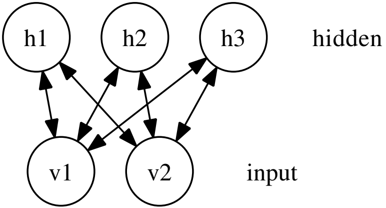
Restricted Boltzmann Machine - **unsupervised** learning machine:



Restricted Boltzmann Machine (RBM)

- A simple **unsupervised** learning module (with no softmax output);
- Only one layer of hidden units and one layer of input units;
- No connection between hidden units (i.e. a special case of Boltzmann Machine);
- Edges are undirected or bi-directional

e.g., an RBM with 2 visible and 3 hidden units:



DEMO:

RBM learning unlabelled hand-written digits

Objective function of RBM - maximum likelihood:

$$E(\mathbf{v},\mathbf{h}|\theta) = \sum_{ij} w_{ij} v_i h_j + \sum_i b_i v_i + \sum_j b_j h_j$$

$$p(\mathbf{v}|\theta) = \prod_{n=1}^{N} \sum_{\mathbf{h}} p(\mathbf{v},\mathbf{h}|\theta) = \prod_{n=1}^{N} \frac{\sum_{\mathbf{h}} \exp(-E(\mathbf{v},\mathbf{h}|\theta))}{\sum_{\mathbf{v},\mathbf{h}} \exp(-E(\mathbf{v},\mathbf{h}|\theta))}$$

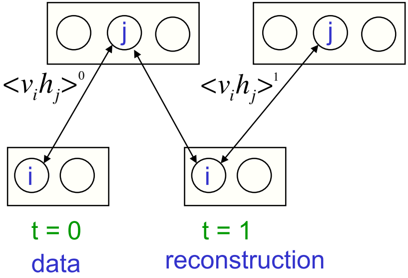
$$\log p(\mathbf{v}|\theta) = \sum_{n=1}^{N} \left(\log \sum_{\mathbf{h}} \exp(-E(\mathbf{v},\mathbf{h}|\theta)) - \log \sum_{\mathbf{v},\mathbf{h}} \exp(-E(\mathbf{v},\mathbf{h}|\theta)) \right)$$

$$\frac{\partial \log p(\mathbf{v}|\theta)}{\partial w_{ij}} = \sum_{n=1}^{N} \left[v_i \sum_{\mathbf{h}} h_j p(\mathbf{h}|\mathbf{v}) - \sum_{\mathbf{v},\mathbf{h}} v_i h_j p(\mathbf{v},\mathbf{h}) \right]$$

$$= \mathbb{E}_{\text{data}}[v_i h_j] - \mathbb{E}_{\text{model}}[\hat{v}_i \hat{h}_j] \equiv < v_i h_j >_{\text{data}} - < \hat{v}_i \hat{h}_j >_{\text{model}}$$

But $< \hat{v}_i \hat{h}_j >_{\text{model}}$ is still too large to estimate, we apply Markov Chain Monte Carlo (MCMC) (i.e., Gibbs sampling) to estimate it.

How Gibbs sampling works



1. Start with a training vector on the visible units
2. Update all the hidden units in parallel
3. Update all the visible units in parallel to get a "reconstruction"
4. Update the hidden units again

$$\Delta w_{ij} = \epsilon(< v_i^0 h_j^0 > - < v_i^1 h_j^1 >) \tag{1}$$

Approximate maximum likelihood learning

$$\frac{\partial \log p(\mathbf{v})}{\partial w_{ij}} \approx \frac{1}{N} \sum_{n=1}^{N} \left[v_i^{(n)} h_j^{(n)} - \hat{v}_i^{(n)} \hat{h}_j^{(n)} \right] \tag{2}$$

where

- $v_i^{(n)}$ is the value of i^{th} visible (input) unit for n^{th} training case;
- $h_j^{(n)}$ is the value of j^{th} hidden unit;
- $\hat{v}_i^{(n)}$ is the **sampled** value for the i^{th} visible unit or the **negative data** generated based on $h_j^{(n)}$ and w_{ij};
- $\hat{h}_i^{(n)}$ is the **sampled** value for the j^{th} hidden unit or the **negative hidden activities** generated based on $\hat{v}_i^{(n)}$ and w_{ij};

Still how exactly the **negative data** and **negative hidden activities** are generated?

wake-sleep algorithm (Lec18 p5)

1. Positive ("wake") phase (clamp the visible units with data):
   - Use input data to generate hidden activities:

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

   Sample hidden state from Bernoulli distribution:

$$h_j \leftarrow \begin{cases} 1, & \text{if } h_j > \texttt{rand(0,1)} \\ 0, & \text{otherwise} \end{cases}$$

2. Negative ("sleep") phase (unclamp the visible units from data):
   - Use h_j to generate **negative data**:

$$\hat{v}_i = \frac{1}{1 + \exp(-\sum_j w_{ij} h_j - b_i)}$$

   - Use negative data $\hat{v}_i$ to generate **negative hidden activities**:

$$\hat{h}_j = \frac{1}{1 + \exp(-\sum_i \hat{v}_i w_{ij} - b_j)}$$

wake-sleep algorithm (con'td) - Learning

$$\Delta w_{ij}^{(t)} = \eta \Delta w_{ij}^{(t-1)} + \epsilon_w \left(\frac{\partial \log p(v|\theta)}{\partial w_{ij}} - \lambda w_{ij}^{(t-1)} \right)$$

$$\Delta b_i^{(t)} = \eta \Delta b_i^{(t-1)} + \epsilon_{vb} \frac{\partial \log p(v|\theta)}{\partial b_i}$$

$$\Delta b_j^{(t)} = \eta \Delta b_j^{(t-1)} + \epsilon_{hb} \frac{\partial \log p(v|\theta)}{\partial b_j}$$

where

$$\frac{\partial \log p(v|\theta)}{\partial w_{ij}} \approx \frac{1}{N} \sum_{n=1}^{N} \left[v_i^{(n)} h_j^{(n)} - \hat{v}_i^{(n)} \hat{h}_j^{(n)} \right]$$

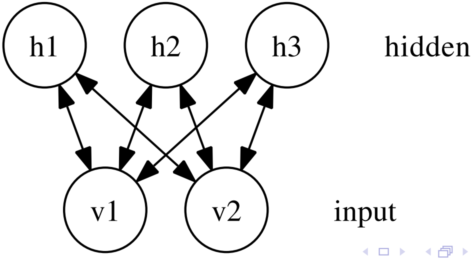
$$\frac{\partial \log p(v|\theta)}{\partial b_i} \approx \frac{1}{N} \sum_{n=1}^{N} \left[v_i^{(n)} - \hat{v}_i^{(n)} \right]$$

$$\frac{\partial \log p(v|\theta)}{\partial b_j} \approx \frac{1}{N} \sum_{n=1}^{N} \left[h_j^{(n)} - \hat{h}_j^{(n)} \right]$$

Match with the matlab code in rbmfun.m in A4 handout.

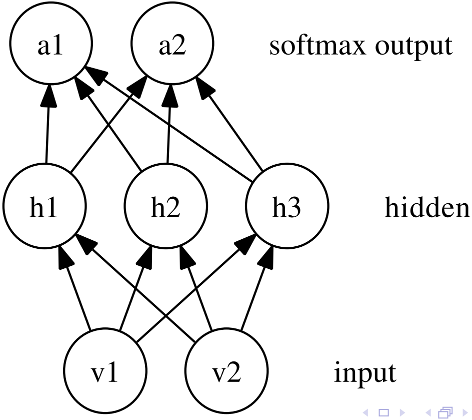
Assignment 4: Initialize backpropagation with RBM

1. First apply RBM to find a sensible set of weights using unlabelled data.



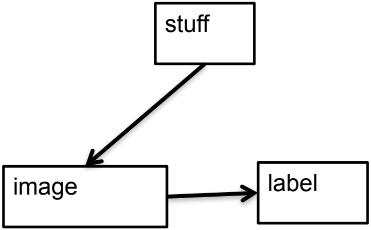
Assignment 4: Initialize backpropagation with RBM

1. First apply RBM to find a sensible set of weights using unlabelled data.
2. Then use the pre-trained weight to perform backpropagation to classify labelled data

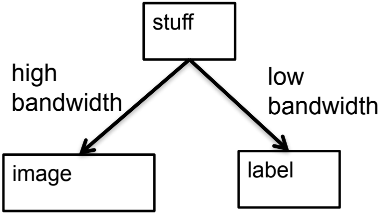


Two plausible ways of cognitive learning (Lecture 21 p9)

Supervised learning (backprop) without unsupervised pre-training (RBM)



Supervised learning (backprop) with unsupervised pre-training (RBM)



ASSIGNMENT 4 DESCRIPTION

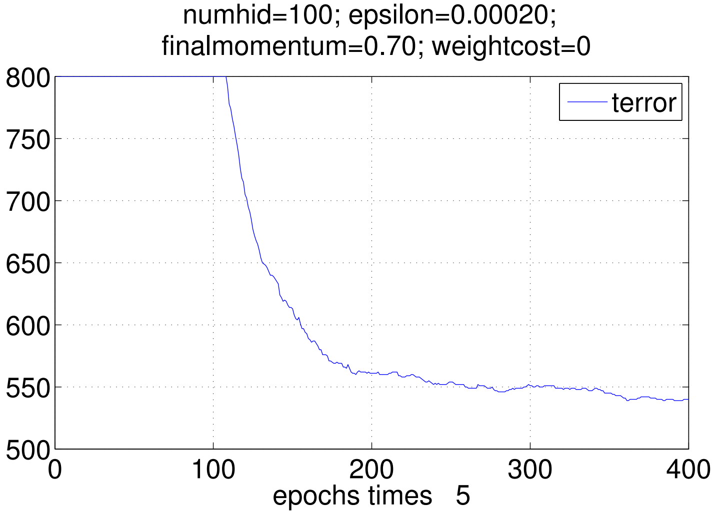
PRELIMINARIES

- First, from Assignment 2, copy the files assign2data2012.mat, classbp2.m and show.m. Recall that the first file has 3000 test and 150 training data points of digits. The second file implements a feedforward neural network and uses the backpropagation algorithm to learn the weights of the neural network from training data.
- Second, copy and unzip the following archive:

  `http://www.cs.toronto.edu/~bonner/courses/2014s/csc321/assignments/hw4_matlab.zip`

- Run train_nn.m in Matlab to train a neural network on the 150 training cases and test on the 3000 test cases. How many errors do you see at the end of the run ?

Recall: in A2, using the 3000 test/validation and 150 training cases, we found the best numhid, weightcost, epsilon, and finalmomentum with the following setting that produced < 550 terror in 2000 epochs. Can we do better than that?



ASSIGNMENT 4

- You will use the other files in the archive to train an RBM, which will be used to initialize (pretrain) the neural network for predicting digits.
- The file unlabeled.mat contains unlabeled training data. You can load the data by using the command "`load unlabeled`" in Matlab. The file rbmfun.m contains code to train an rbm, and the file showrbmweights.m allows you to visualize the weights of the rbm.
- The point of the assignment is to figure out how to use the 2000 unlabeled cases and the function rbmfun to do better on the test set (which should really be called a validation set since you use it many times for deciding things like the number of hidden units).

PART 1. (5 points)

- Using results from running rbmfun on the unlabeled data, modify classbp2 in a way that allows you to get **a best test error of less than 500 in at least 5 runs out of 10, and less than 490 in at least one of these runs.** What are the exact changes you made? Give a list of the variables and the values you assigned to them that allowed you to get these results.
- If you cannot achieve the desired error rate, give the exact details of the best settings you could find.
- Also report the error that you get with the same settings for classbp2 but without using rbmfun and the unlabeled data.

Suggestions for Part 1

1. classbp2FineTuneVISHID: in class2bp.m, under `if restart==1`:
   - change `inhid = initialweightsize*randn(numin, numhid);` to `inhid = vishid;`
   - comment out `hidbiases = initialweightsize*randn(1, numhid);`
2. Experiment setting for rbmfun, e.g. (u wanna do better):

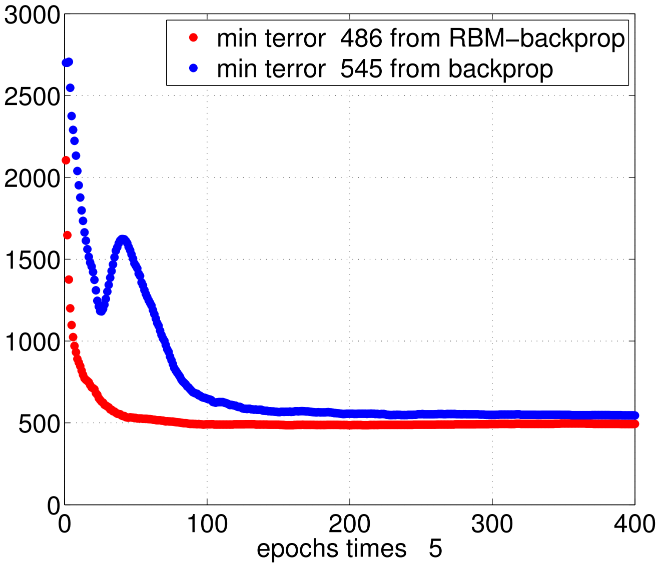
```
rbmMaxepoch = 500;
numhid = 200; % numhid in rbmfun == numhid in class2bp
rbmWeightcost = 0.0002;
```

3. Experiment setting for class2bp, e.g. (u wanna do better):

```
maxepoch = 2000;
epsilon = 10^-4;
finalmomentum = 0.9;
weightcost = 0;
```

4. `[hidbiases, vishid] = rbmfun(unlabeleddata, numhid, rbmWeightcost, rbmMaxepoch);`
5. `restart = 1; classbp2FineTuneVISHID;`

- The above code first trains a RBM with 200 hidden units by setting `vishid` and `hidbiases` for 500 epochs and uses the pre-trained weights to initialize classbp2FineTuneVISHID and do backpropagation for 2000 epochs.
- Run the above setting multiple times, each time record **the best terror** for backpropagation with and without pre-training.
- Experiment with different settings to find even better results than the one in the next two slides.
- You may need to do better than the above setting to get **the best test error less than 500 in at least 5 runs out of 10, and less than 490 in at least one of these runs.**



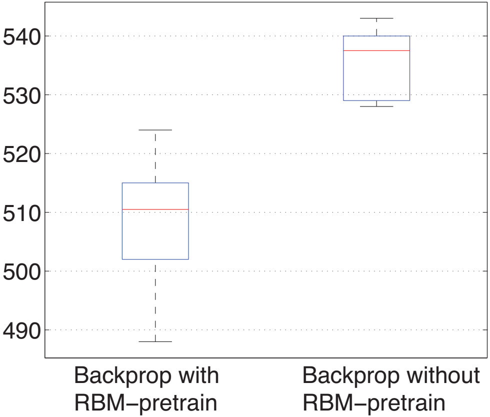
RBM: rbmMaxepoch = 500; numhid = 200; rbmWeightcost = 0.0002;

FFN: maxepoch = 2000; epsilon = 10^-4; finalmomentum = 0.9; weightcost = 0;

Table : Best test errors over 10 runs (again, you may need to do better):

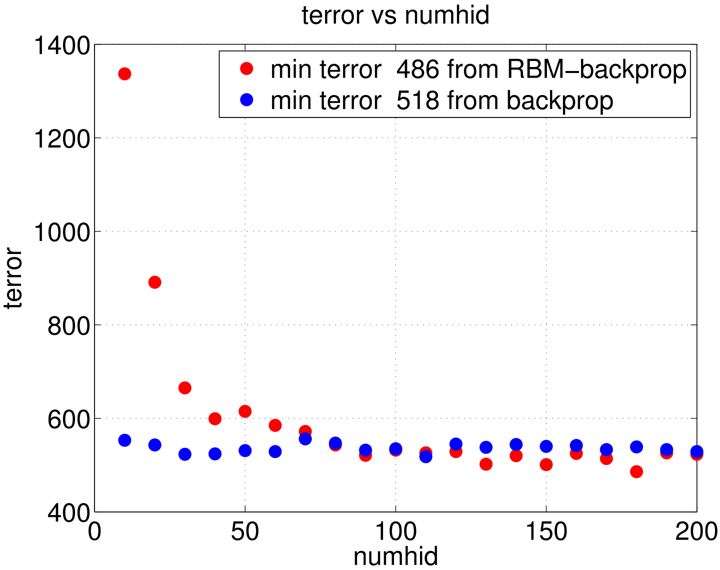
RBM+BP	496	512	510	524	503	488	515	511	517	502
BP	538	539	540	529	543	532	541	537	528	529

Boxplot using the above table



PART 2. (3 points)

- Say how you think the use of the unlabeled data influences the number of hidden units that you should use. Report some evidence for your opinion.
- Suggestion1: experiment with different numhid used in rbmfun and compare the terror.
- Suggestion2: read Page 12 in *A practical guide to training restricted Boltzmann machines* (Hinton, 2010) `http://www.cs.toronto.edu/~hinton/absps/guideTR.pdf`.
- Hint: the number of bits that it takes to specify a 16×16 image is much higher than the bits used to specify the corresponding 1-of-10 label.



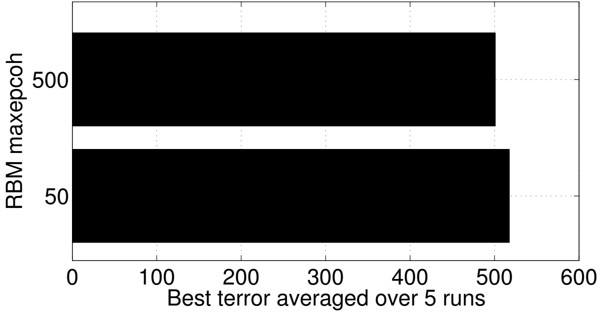
RBM: rbmMaxepoch = 100; **numhid = 10:10:200**; rbmWeightcost = 0.0002; FFN: maxepoch = 2000; epsilon = 10^-4; finalmomentum = 0.9; weightcost = 0.

PART 3. (2 points)

- Using the same parameters (other than maxepoch) as in the file train_nn.m, run your experiment by training the rbmfun for **50** and **500** epochs and report the best test set error for each case (averaged over five runs).
- On the basis of these numbers what can you say about the effect of number of epochs of RBM training on the final test error from the neural network?

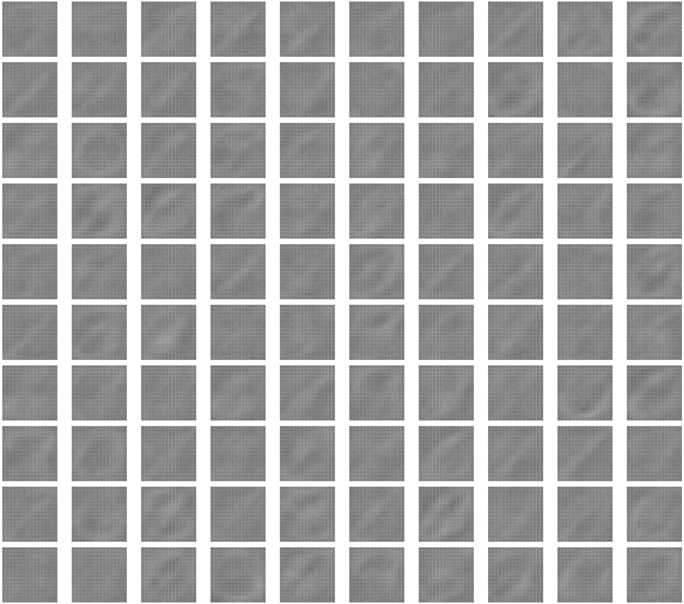
Your report should not be more than three pages long (including figures) and should not contain more than 2 pages of text. One page of text is quite sufficient.

Avg best error: 50 rbmMaxepoch: 517.6; 500 rbmMaxepoch: 501.0



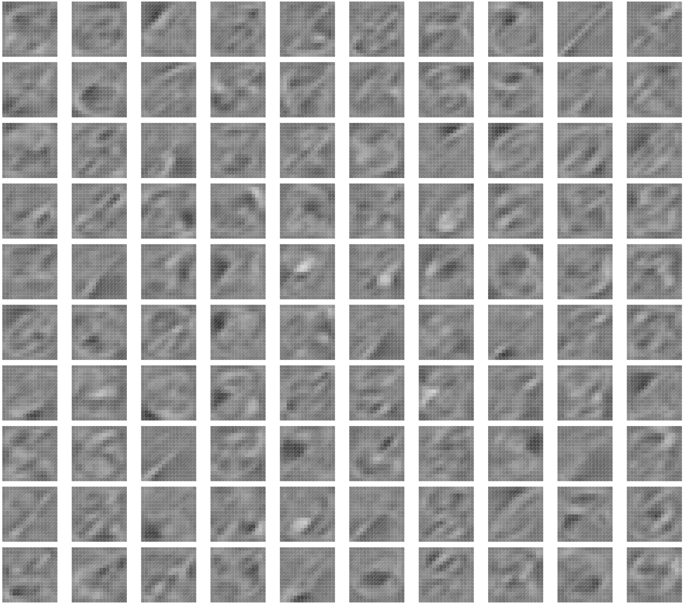
Setting:
RBM: **rbmMaxepoch = 50 or 500**; numhid = 100; rbmWeightcost = 0;
FFN: maxepoch = 2000; epsilon = 0.01; finalmomentum = 0.8; weightcost = 0.

NB: Longer training in RBM amounts to longer unsupervised learning time of the underlying features of the images



RBM: **maxepoch = 50**; numhid = 100; rbmWeightcost = 0;

FFN: maxepoch = 2000; epsilon = 0.01; finalmomentum = 0.8; weightcost = 0.



RBM: **maxepoch = 500**; numhid = 100; rbmWeightcost = 0;

FFN: maxepoch = 2000; epsilon = 0.01; finalmomentum = 0.8; weightcost = 0.