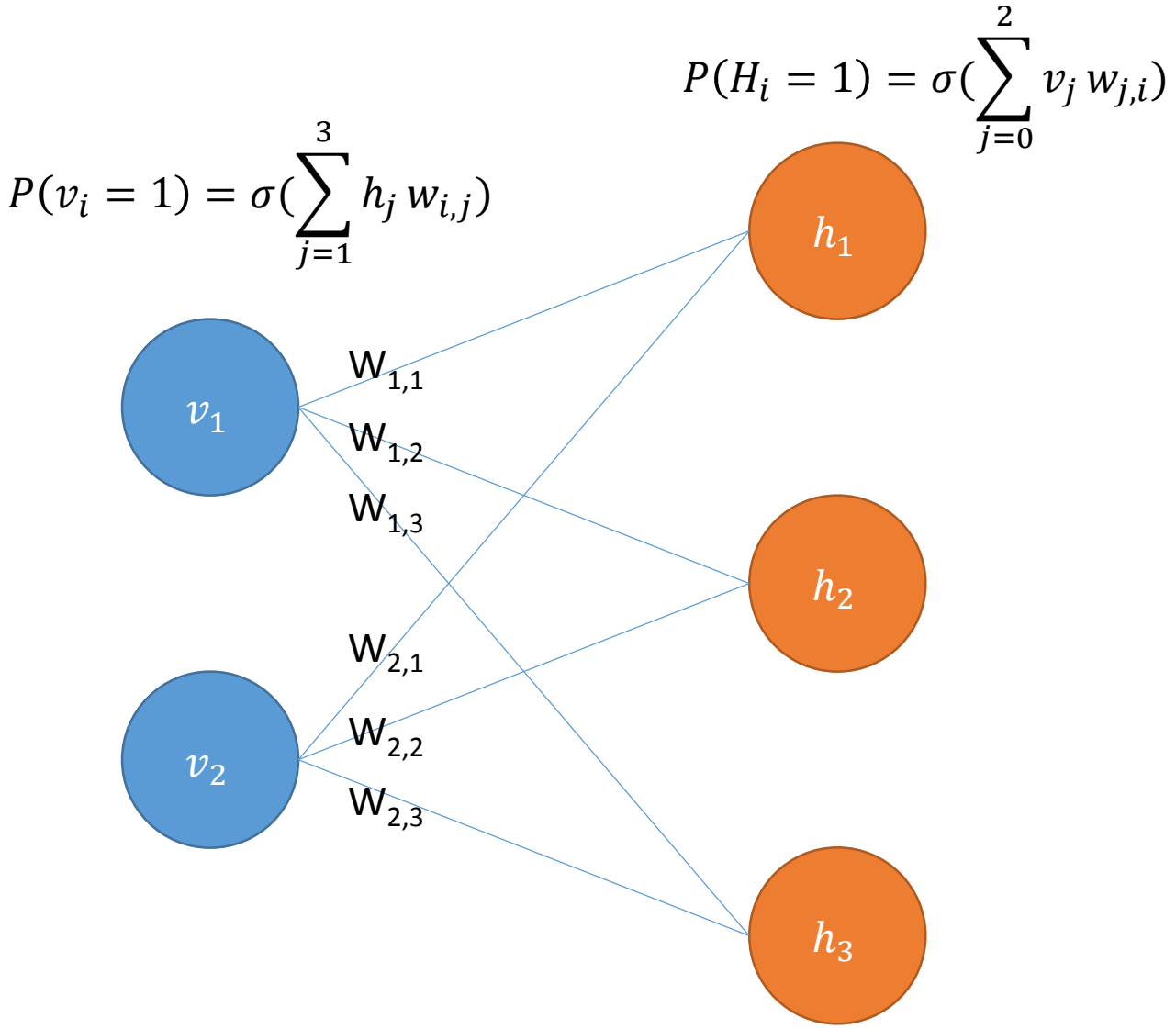
Graphical Models II

(Binary) Restricted Boltzmann Machines

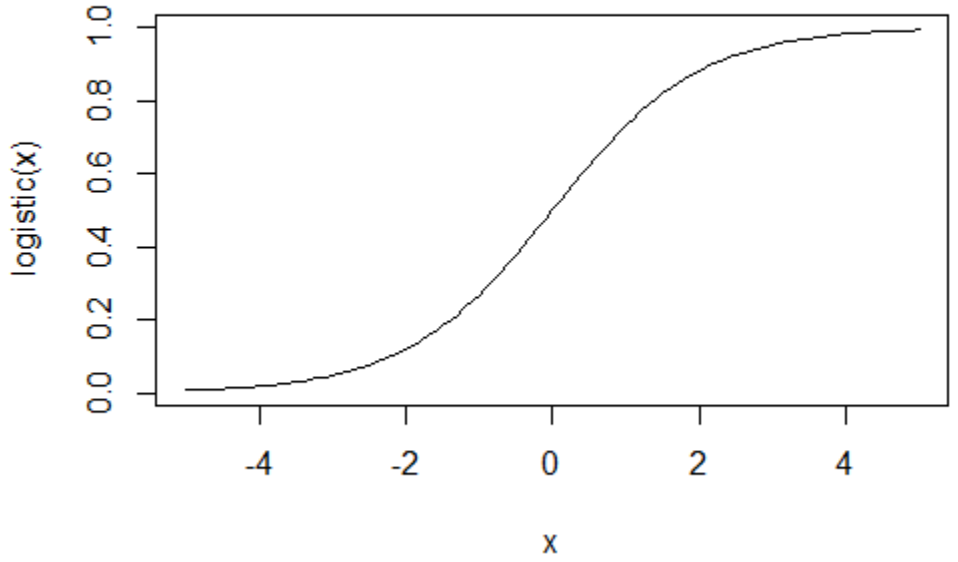
- Visible units:
  - Input variables: $x_1, \ldots, x_n$
- Hidden units
- Often add a bias unit, but we ignore this.
- Weights on edges are (shared) parameters in (two) conditional distributions.



Restricted Boltzmann Machines

- σ is the standard logistic function:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



Restricted Boltzmann Machines

- Inference
  - As in any Markov Random Field: Gibbs Sampling ☺
  - Set known variables to known values, perform Gibbs Sampling to find probability distributions for unknown variables.
- Training

  Set initial weights randomly. Hinton: small random values $\mathcal{N}(0, .01)$.

  For each datum in a dataset:

  Compare an unbiased sample of the state of the model given a datum with an unbiased sample of the state of the free model (without any data). Adjust the weights so that the latter becomes closer to the former:

  $$\Delta w = \epsilon(\langle vh \rangle_{data} - \langle vh \rangle_{model})$$

  Where ϵ is the learning rate and $\langle x_i x_j \rangle$ is the outer product.

  Repeat until convergence or as desired.

Restricted Boltzmann Machine

Problem:

- Easy to get an unbiased sample of the state of the model given a datum. Since the hidden variable are conditionally independent given the visable variables, calculate the probability distribution of the hidden variables exactly and then sample!
- Hard to get an unbiased sample of the free model. We could run Gibbs Sampling with no known variables, but this is very time consuming.

Contrastive Divergence

Solution

- We don't actually need a very accurate sample from the free model – just enough to let us know which way to update the weights.
- Let's just run Gibbs Sampling a small number, *n*, of steps.
- We can start from any random state. Why not start from the datum? Then we will get samples closer to being from the free that are '*from the direction*' of the unbiased sample constrained by the datum.

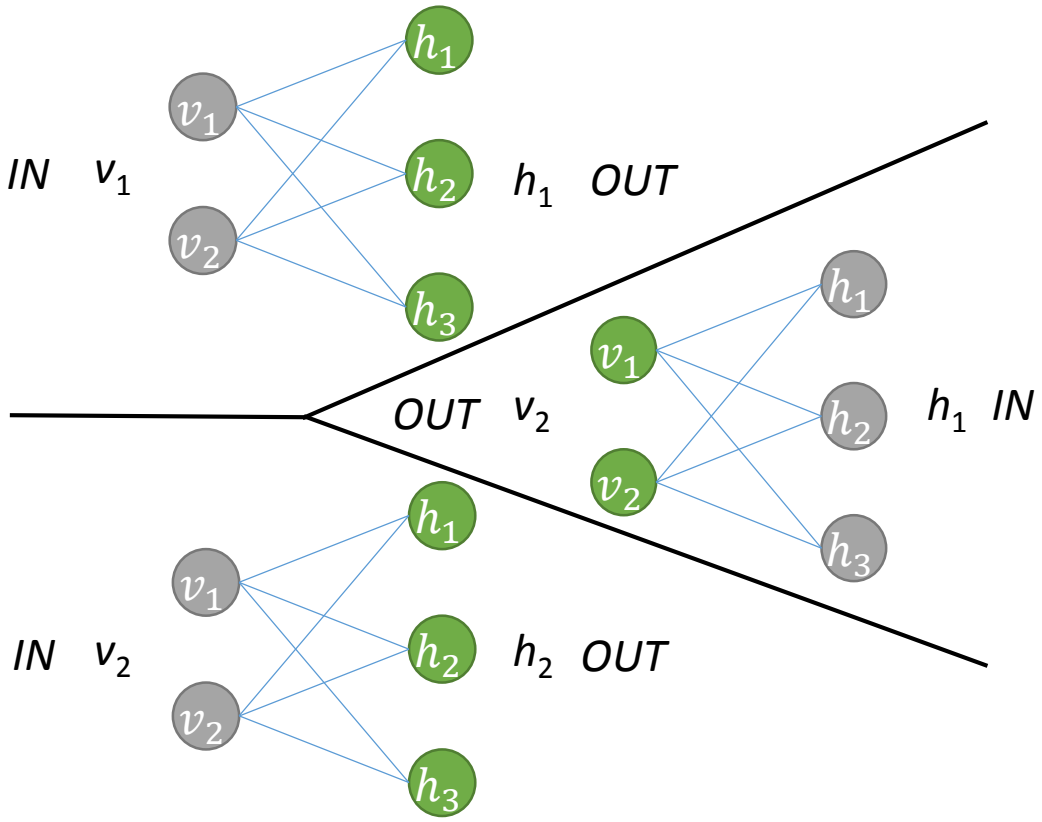
This is called the Contrastive Divergence *n* algorithm. Often *n*=1.

(This is a naive version – improvements are common.)

Contrastive Divergence

CD_1 algorithm:

1. Sample values for all hidden units given the visible units = datum = $v_1 : h_1$.
2. Sample new values for the visible units given $h_1 : v_2$.
3. Sample new values for the hidden units given $v_2 : h_2$.



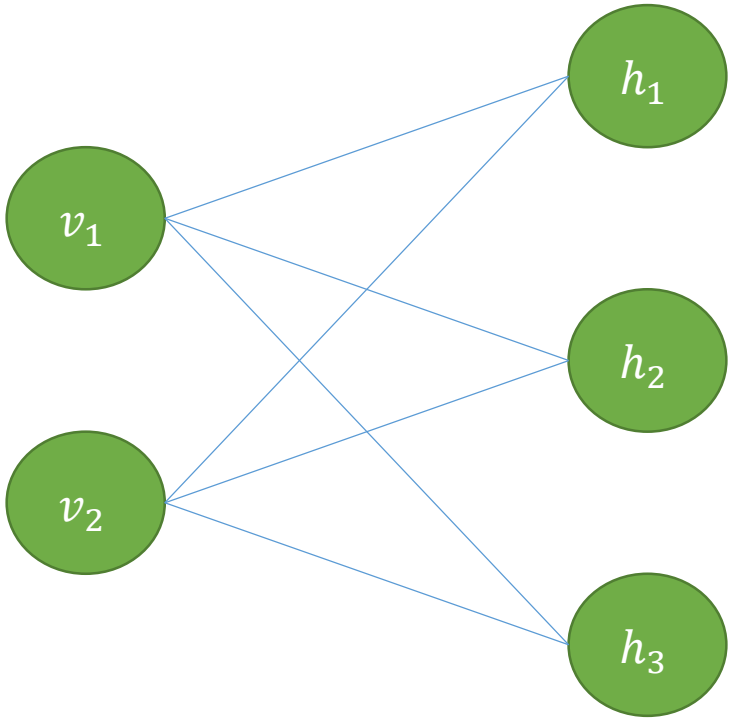
- Problem:

Train a RBM with 2 visible and 3 hidden variables from the data:

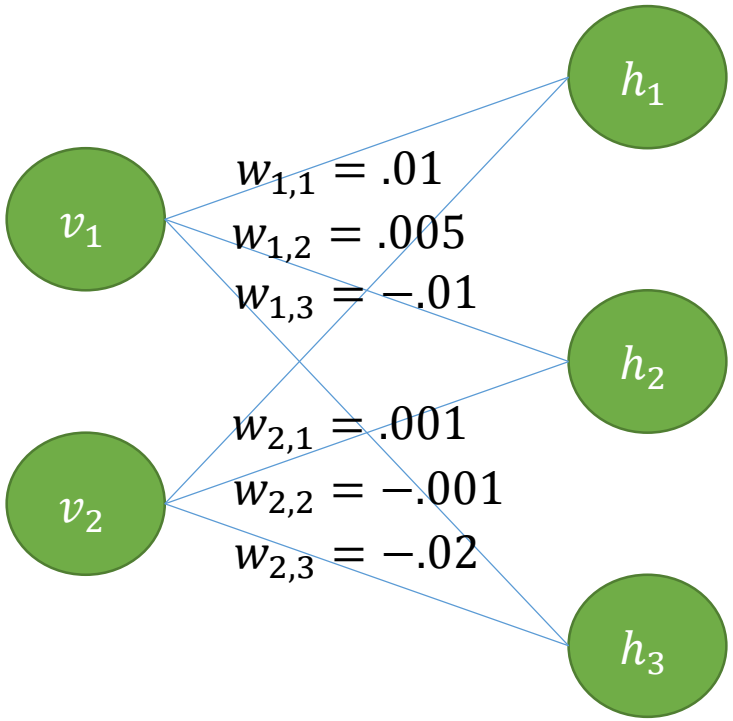
X1	X2
1	0
1	1
…	…

Let the learning rate, ϵ, be 0.01.

Set up the system.



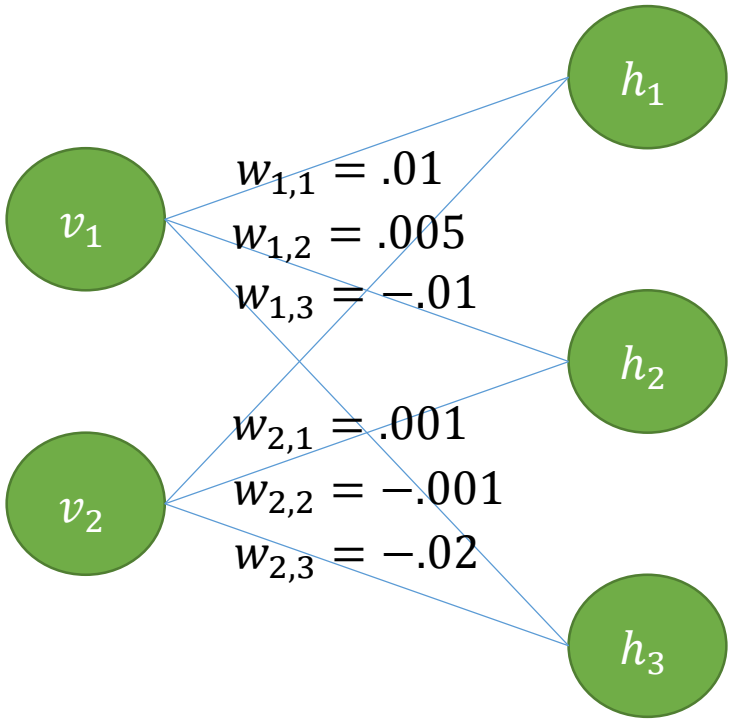
Assign initial random weights from $\mathcal{N}(0, .01)$.



X1	X2
1	0
1	1
…	…

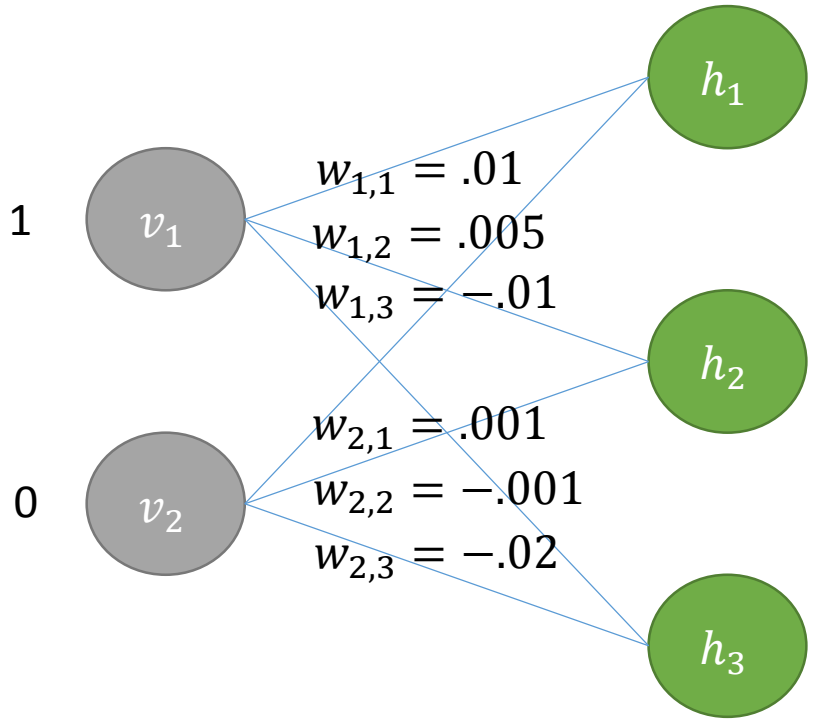
Begin training. First datum=<1,0>

$V_1 = ?$
$H_1 = ?$
$V_2 = ?$
$H_2 = ?$



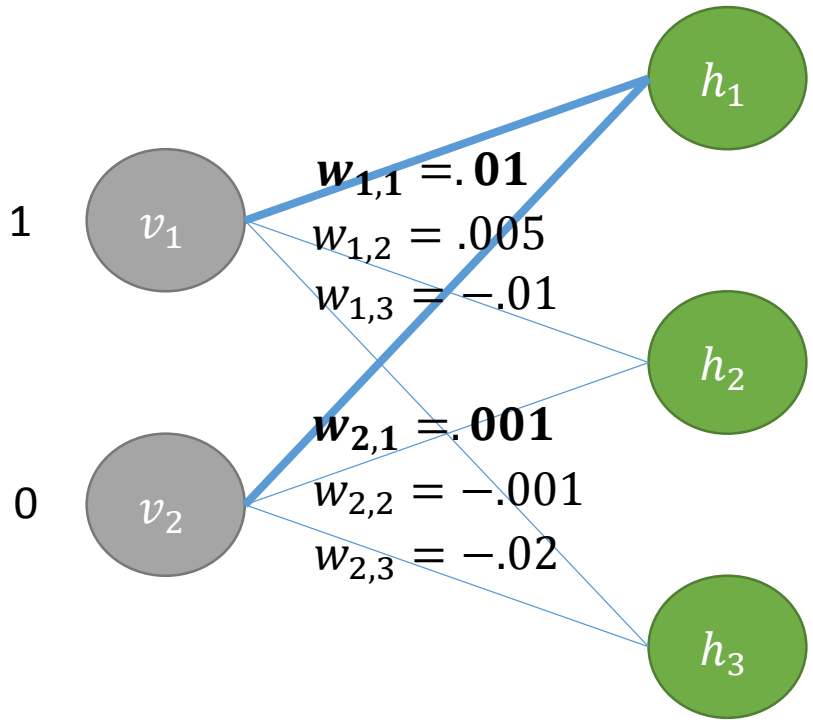
We know $V_1 =< 1,0 >$. To find H_1 we start by setting $v_1 = 1$ and $v_2 = 0$

$V_1 =$ <1,0>
$H_1 = ?$
$V_2 = ?$
$H_2 = ?$



Now we work out the probability distribution for each hidden item given V_1.

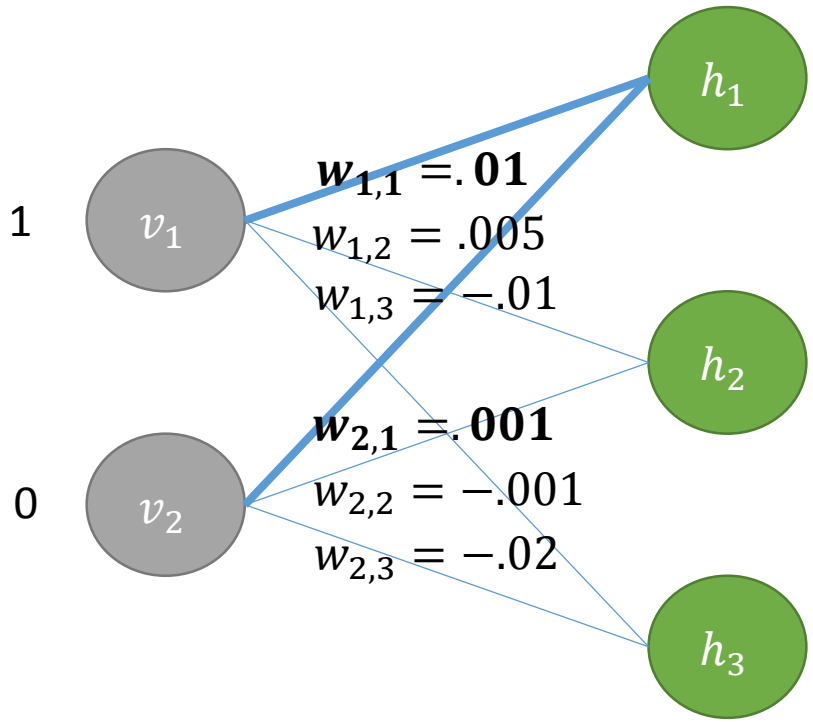
$V_1 = <1,0>$
$H_1 = ?$
$V_2 = ?$
$H_2 = ?$



$$P(h_1 = 1) = \sigma(\sum_{j=0}^{2} v_j \, w_{j,1})$$

Now we work out the probability distribution for each hidden item given V_1.

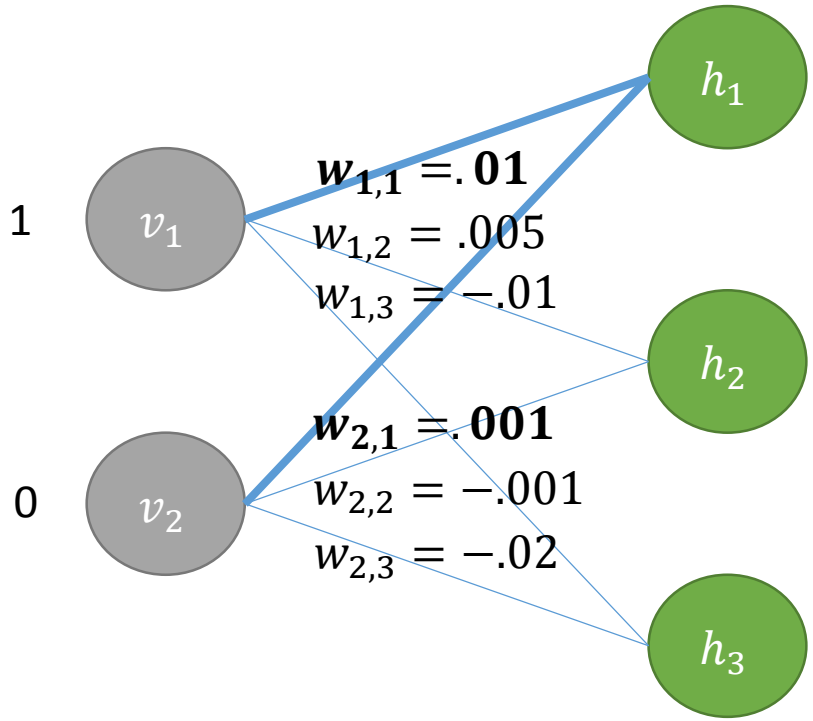
$V_1 = <1,0>$
$H_1 = ?$
$V_2 = ?$
$H_2 = ?$



$$P(h_1 = 1) = \sigma\left(\sum_{j=0}^{2} v_j\, w_{j,1}\right)$$
$$= \sigma\big((1)(.01) + (0)(.001)\big)$$
$$= \sigma(.01)$$
$$= \frac{1}{1 + e^{-.01}}$$
$$= .5025$$

Now we work out the probability distribution for each hidden item given V_1.

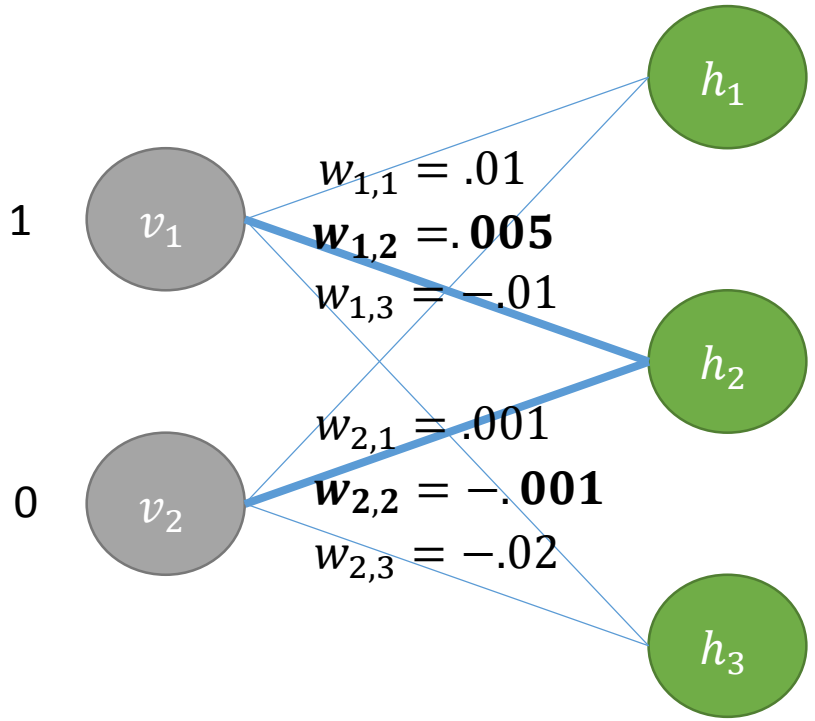
$V_1 = <1,0>$
$H_1 = ?$
$V_2 = ?$
$H_2 = ?$



$P(h_1 = 1) = .5025$

Now we work out the probability distribution for each hidden item given V_1.

$V_1 = <1,0>$
$H_1 = ?$
$V_2 = ?$
$H_2 = ?$

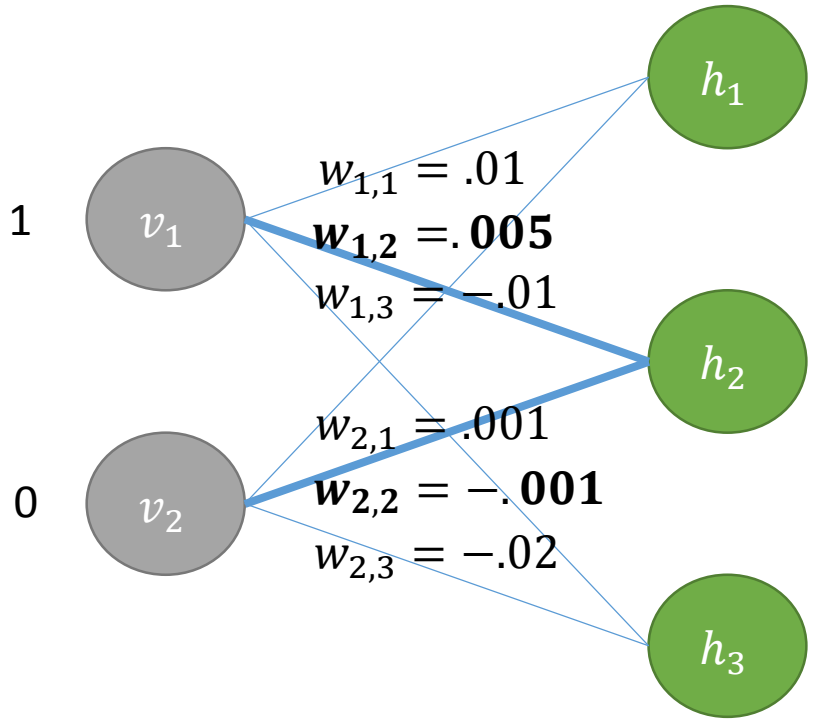


$P(h_1 = 1) = .5025$

$P(h_2 = 1) = \sigma(\sum_{j=0}^{2} v_j \, w_{j,2})$

Now we work out the probability distribution for each hidden item given V_1.

$V_1 = <1,0>$
$H_1 = ?$
$V_2 = ?$
$H_2 = ?$

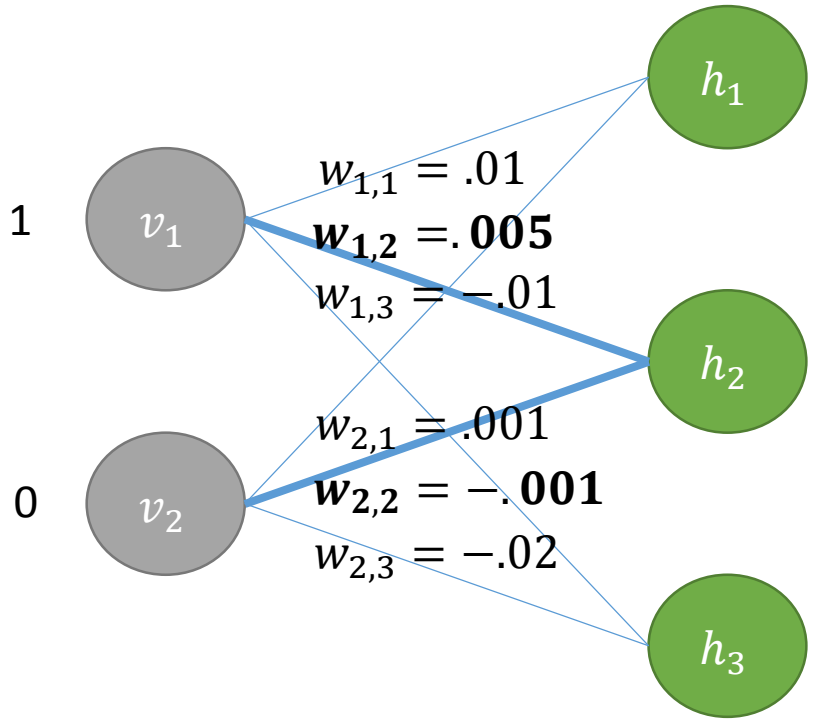


$$P(h_1 = 1) = .5025$$

$$P(h_2 = 1) = \sigma\left(\sum_{j=0}^{2} v_j\, w_{j,2}\right)$$
$$= \sigma\big((1)(.005) + (0)(-.001)\big)$$
$$= \sigma(.005)$$
$$= \frac{1}{1 + e^{-.005}}$$
$$= .5012$$

Now we work out the probability distribution for each hidden item given V_1.

$V_1 = <1,0>$
$H_1 = ?$
$V_2 = ?$
$H_2 = ?$

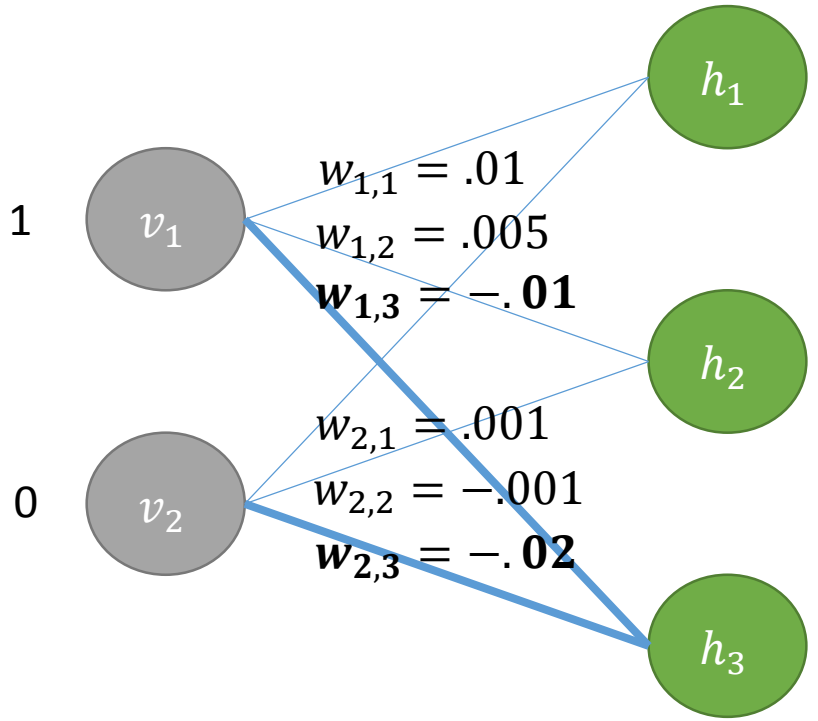


$P(h_1 = 1) = .5025$

$P(h_2 = 1) = .5012$

Now we work out the probability distribution for each hidden item given V_1.

$V_1 = <1,0>$
$H_1 = ?$
$V_2 = ?$
$H_2 = ?$



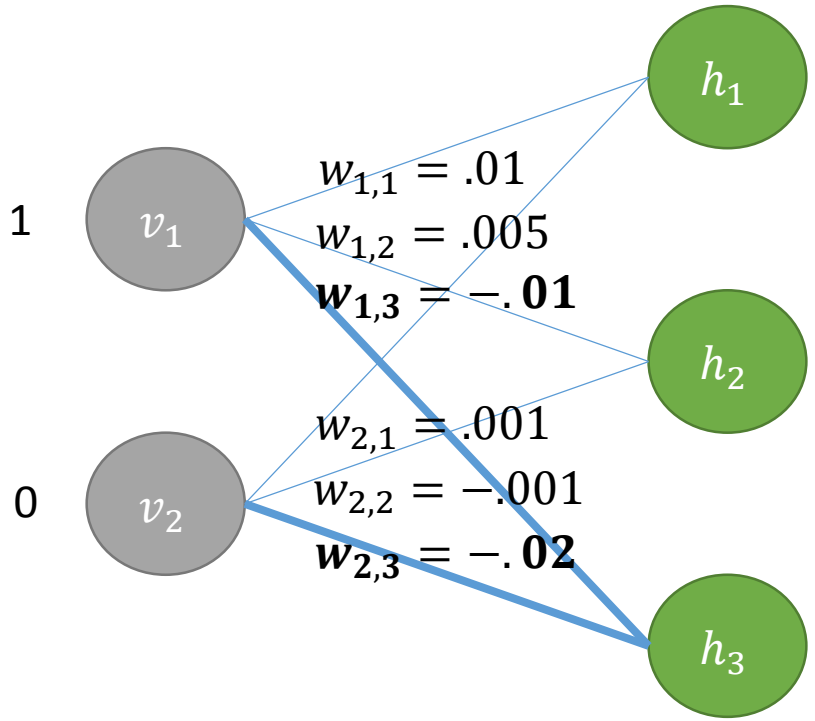
$P(h_1 = 1) = .5025$

$P(h_2 = 1) = .5012$

$P(h_3 = 1) = \sigma(\sum_{j=0}^{2} v_j \, w_{j,3})$

Now we work out the probability distribution for each hidden item given V_1.

$V_1 = <1,0>$
$H_1 = ?$
$V_2 = ?$
$H_2 = ?$



$$P(h_1 = 1) = .5025$$

$$P(h_2 = 1) = .5012$$

$$P(h_3 = 1) = \sigma\left(\sum_{j=0}^{2} v_j\, w_{j,3}\right)$$

$$= \sigma\big((1)(-.01) + (0)(-.02)\big)$$

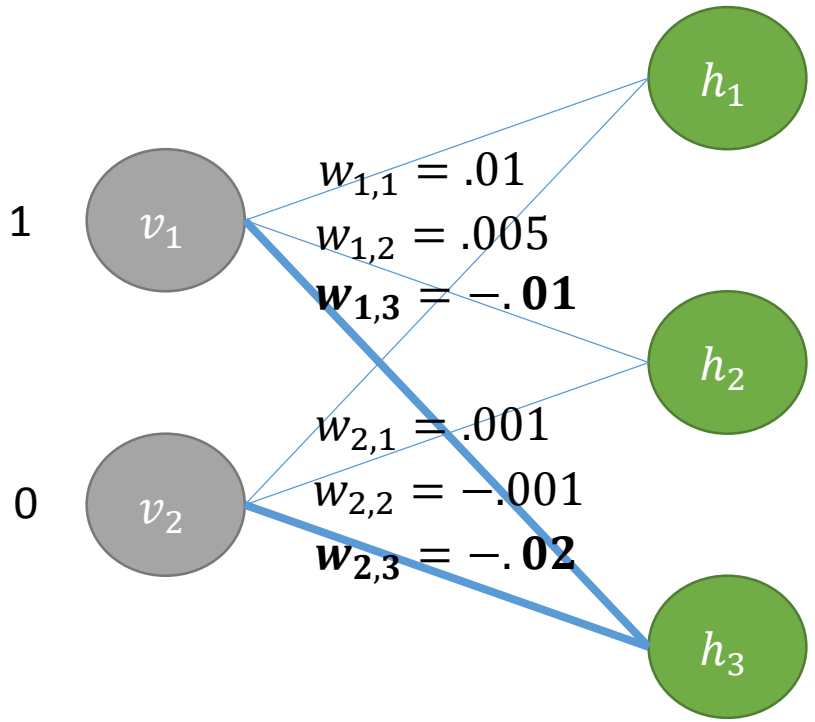
$$= \sigma(-.01)$$

$$= \frac{1}{1 + e^{.01}}$$

$$= .4975$$

Now we work out the probability distribution for each hidden item given V_1.

$V_1 = <1,0>$
$H_1 = ?$
$V_2 = ?$
$H_2 = ?$



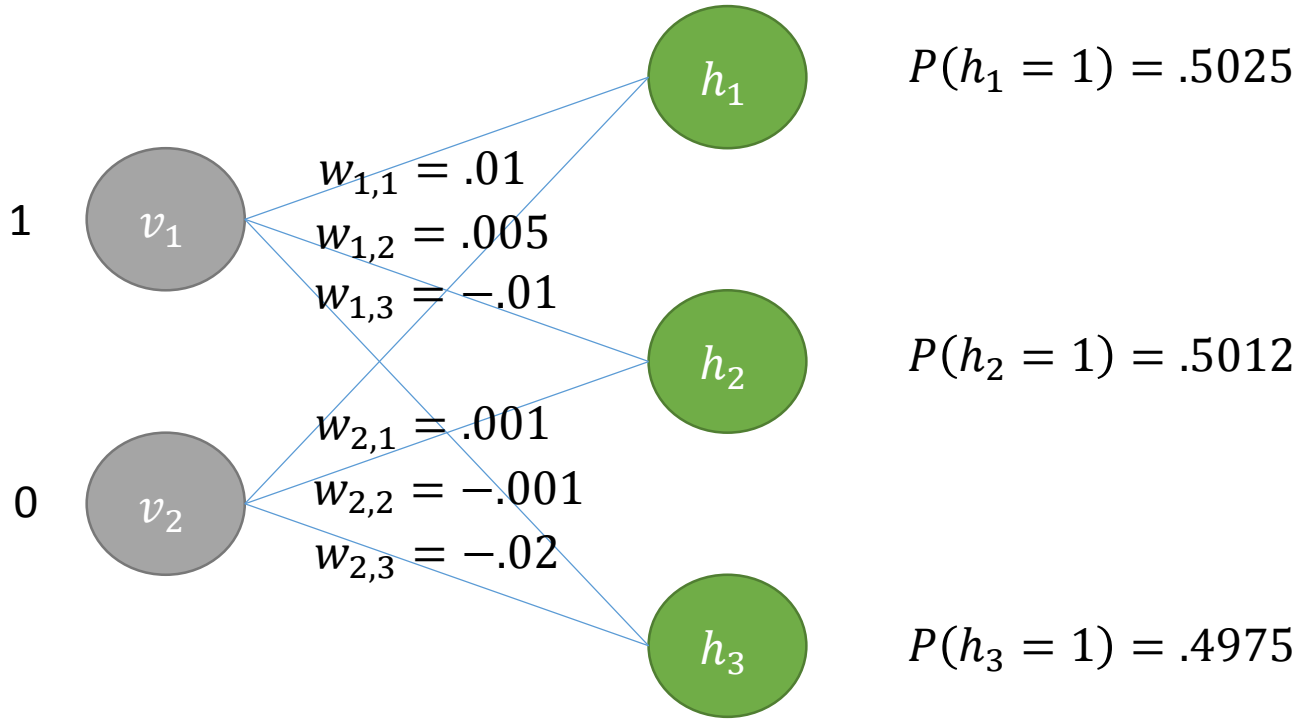
$P(h_1 = 1) = .5025$

$P(h_2 = 1) = .5012$

$P(h_3 = 1) = .4975$

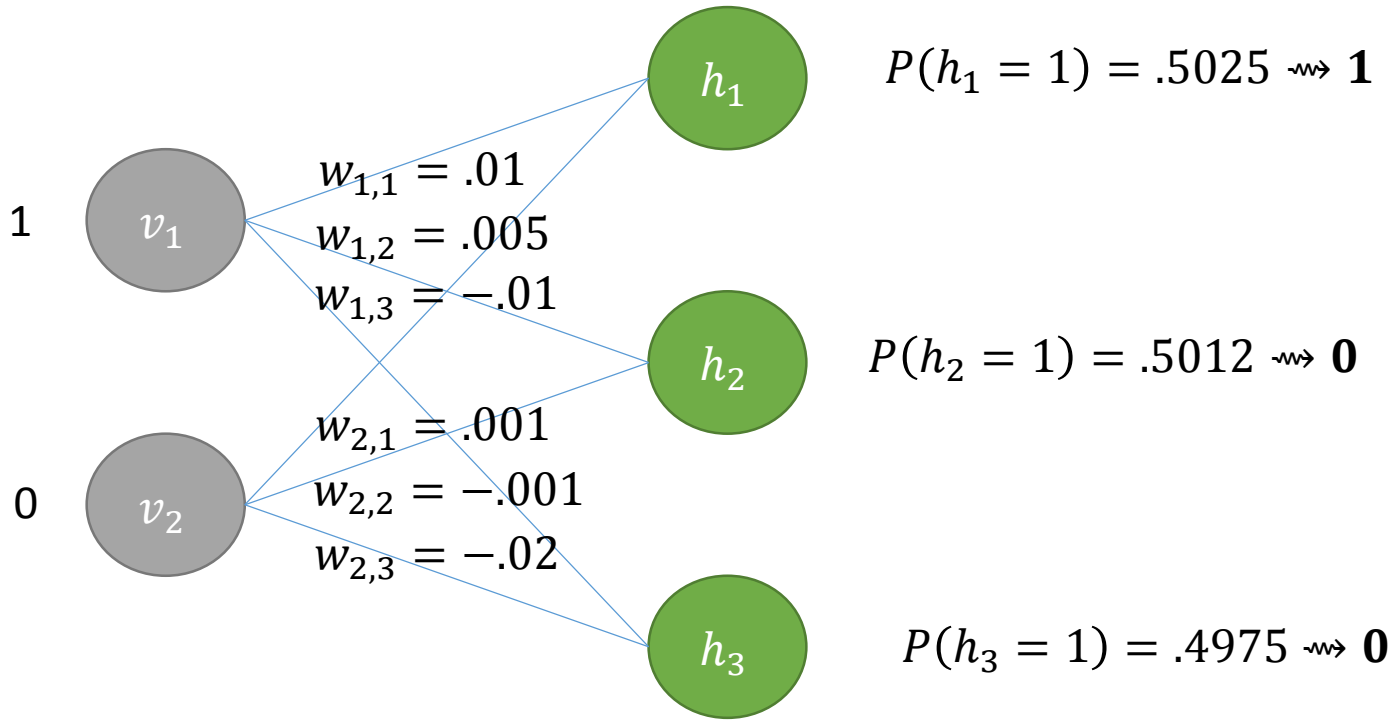
Given these probability distributions we create a random sample for the hidden items.

$V_1 = <1,0>$
$H_1 = ?$
$V_2 = ?$
$H_2 = ?$



Given these probability distributions we create a random sample for the hidden items.

$V_1 = <1,0>$
$H_1 = ?$
$V_2 = ?$
$H_2 = ?$



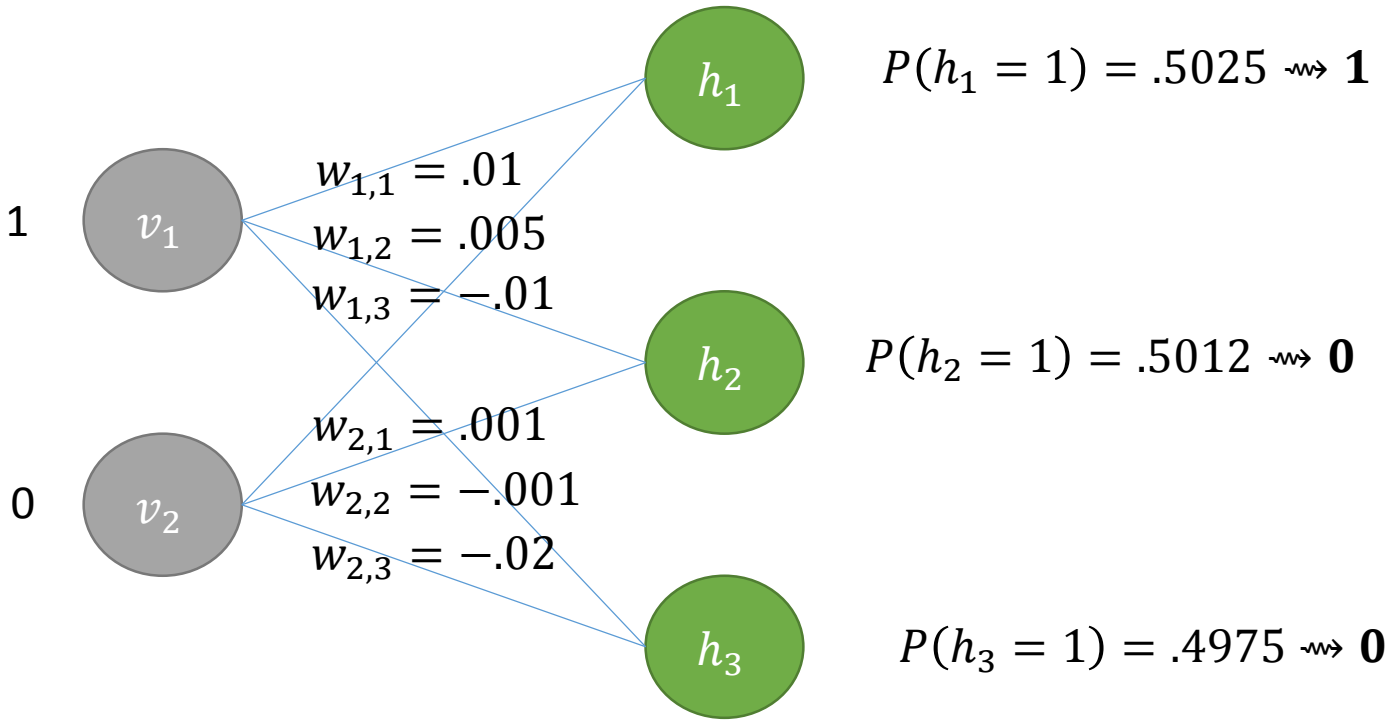
This is H_1.

$V_1 = <1,0>$

$\boldsymbol{H_1 = <1,0,0>}$

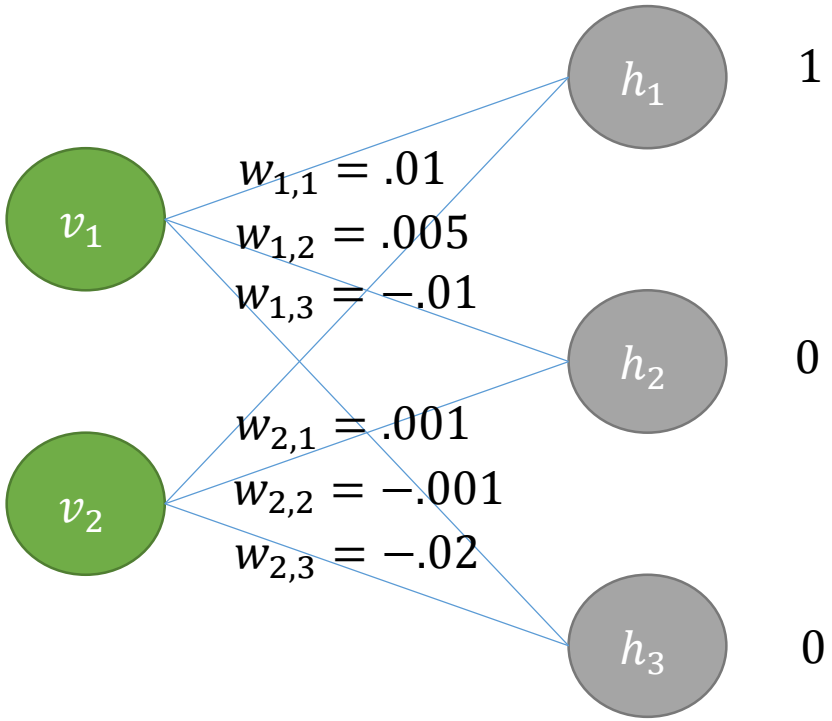
$V_2 = ?$

$H_2 = ?$



Now to find V_2 we 'unlock' the visible variables and 'lock' the hidden variables to H_1.

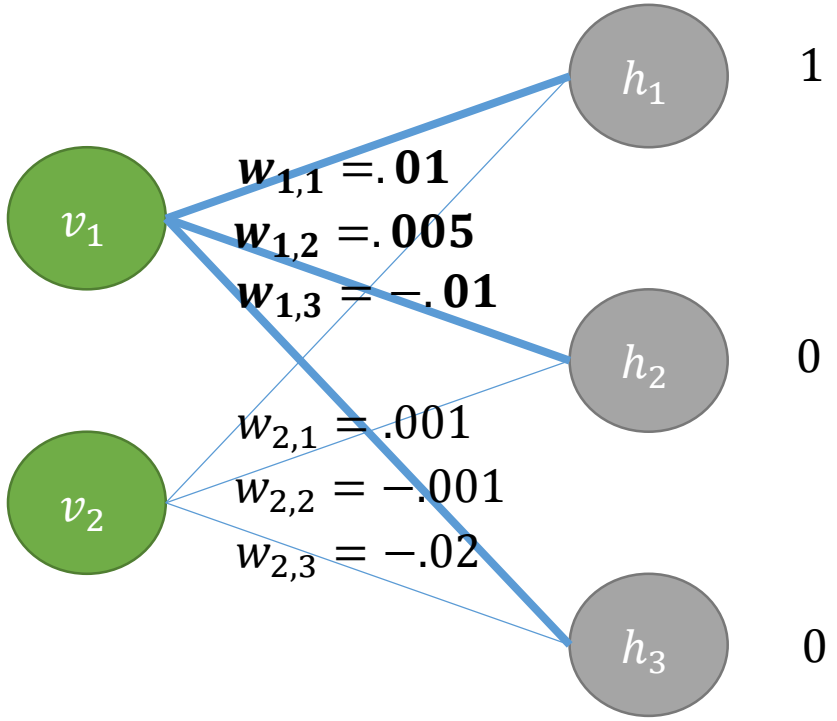
$V_1 =$ <1,0>
$H_1 =$ <1,0,0>
$V_2 = ?$
$H_2 = ?$



Now we work out the probability distribution for each hidden item given H_1.

$V_1 =$ <1,0>
$H_1 =$ <1,0,0>
$V_2 = ?$
$H_2 = ?$

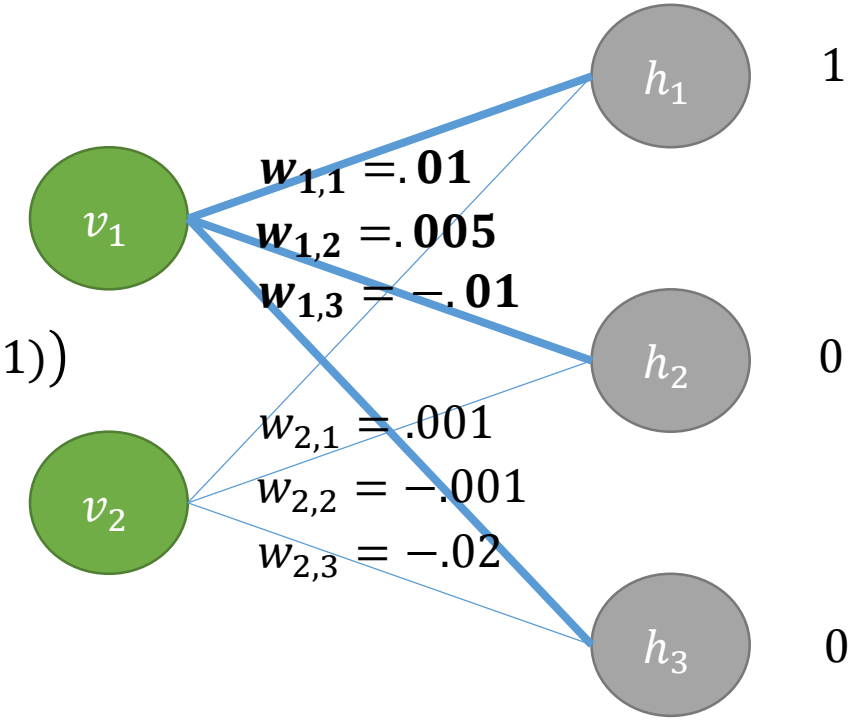
$$P(v_1 = 1) = \sigma(\sum_{j=0}^{2} h_j \, w_{1,j})$$



Now we work out the probability distribution for each hidden item given H_1.

$V_1 = <1,0>$
$H_1 = <1,0,0>$
$V_2 = ?$
$H_2 = ?$

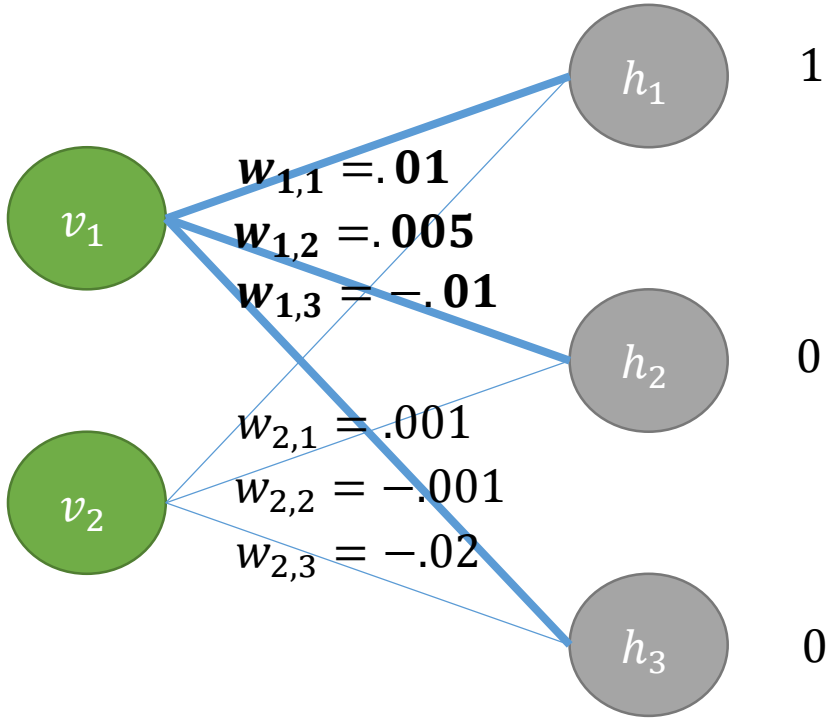
$$P(v_1 = 1) = \sigma\left(\sum_{j=0}^{2} h_j\, w_{1,j}\right)$$
$$= \sigma\big((1)(.01) + (0)(.005) + (0)(-.01)\big)$$
$$= \sigma(.01)$$
$$= \frac{1}{1 + e^{-.01}}$$
$$= .5025$$



Now we work out the probability distribution for each hidden item given H_1.

$V_1 =$ <1,0>
$H_1 =$ <1,0,0>
$V_2 = ?$
$H_2 = ?$

$P(v_1 = 1) = .5025$

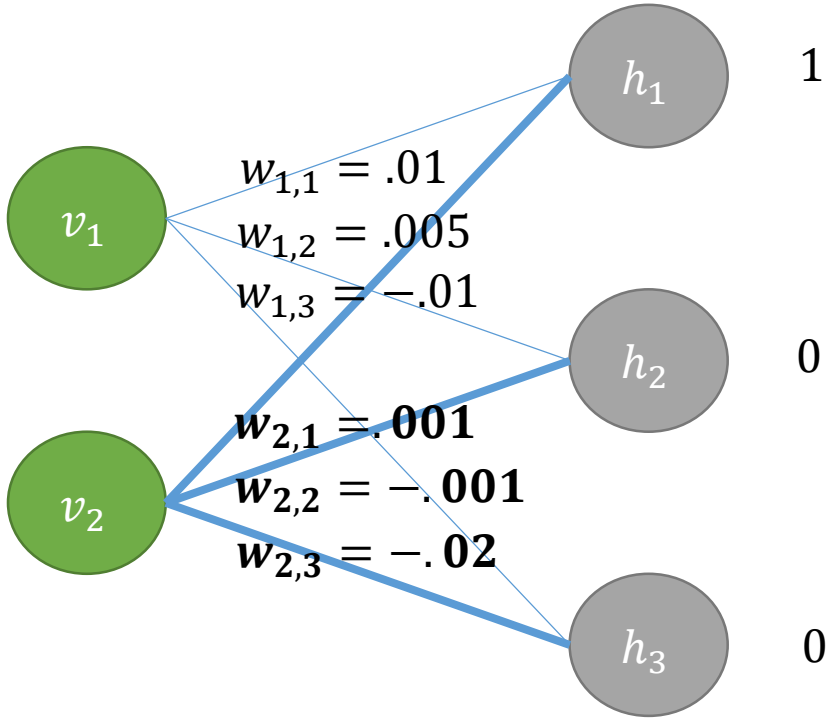


Now we work out the probability distribution for each hidden item given H_1.

$V_1 = <1,0>$
$H_1 = <1,0,0>$
$V_2 = ?$
$H_2 = ?$

$$P(v_1 = 1) = .5025$$

$$P(v_2 = 1) = \sigma(\sum_{j=0}^{2} h_j \, w_{2,j})$$

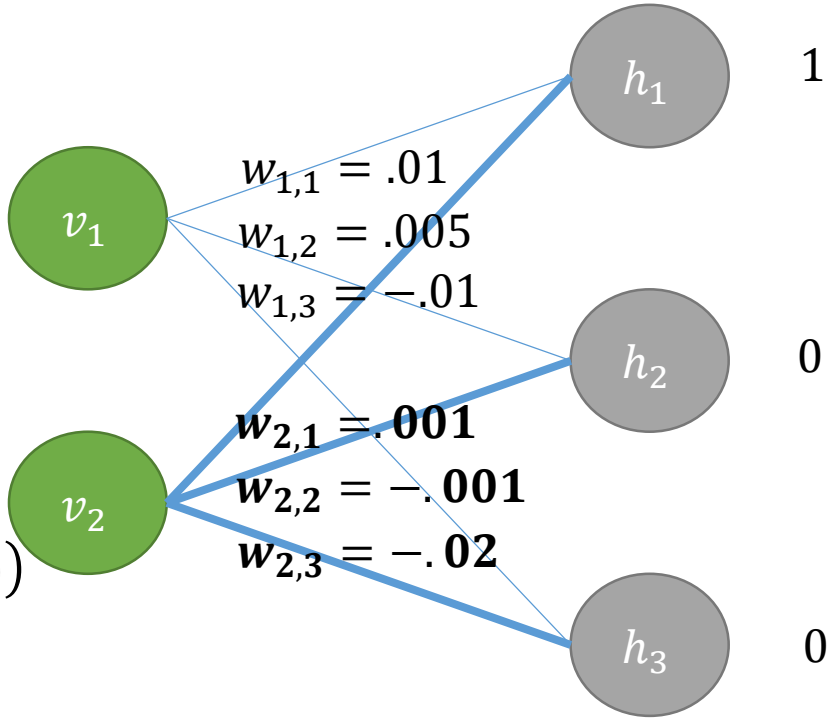


Now we work out the probability distribution for each hidden item given H_1.

$V_1 = <1,0>$
$H_1 = <1,0,0>$
$V_2 = ?$
$H_2 = ?$

$$P(v_1 = 1) = .5025$$

$$P(v_2 = 1) = \sigma\left(\sum_{j=0}^{2} h_j\, w_{2,j}\right)$$
$$= \sigma\big((1)(.001) + (0)(-.001) + (0)(-.02)\big)$$
$$= \sigma(-.001)$$
$$= \frac{1}{1 + e^{.001}}$$
$$= .49975$$

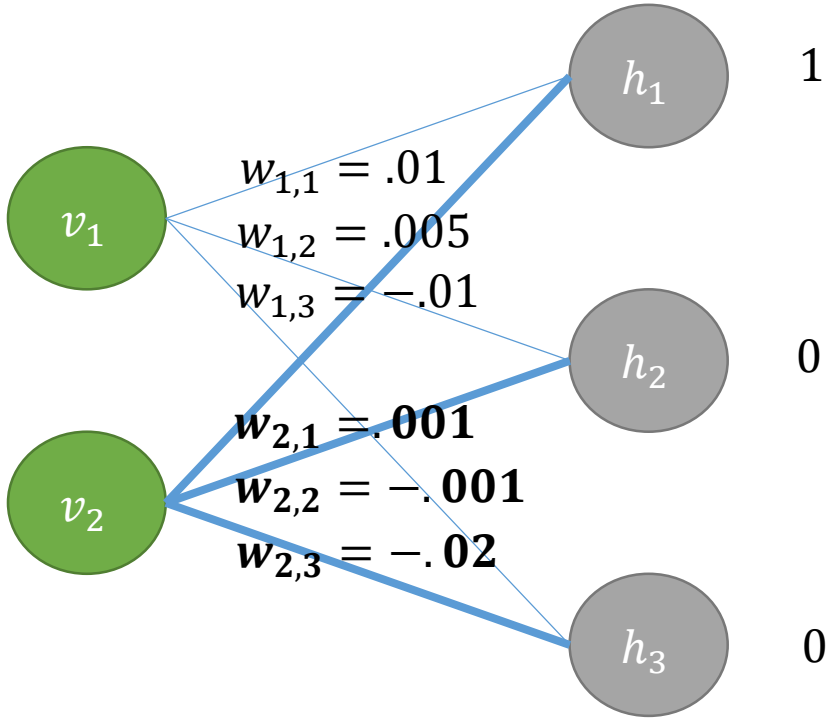


Now we work out the probability distribution for each hidden item given H_1.

$V_1 =$ <1,0>
$H_1 =$ <1,0,0>
$V_2 = ?$
$H_2 = ?$

$P(v_1 = 1) = .5025$

$P(v_2 = 1) = .49975$

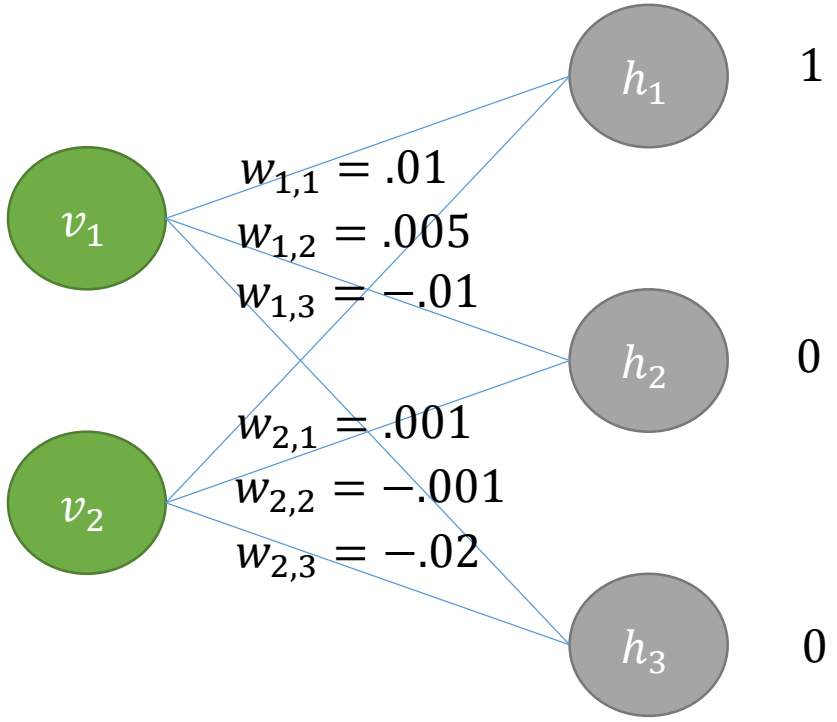


Given these probability distributions we create a random sample for the visible items.

$V_1 = <1,0>$
$H_1 = <1,0,0>$
$V_2 = ?$
$H_2 = ?$

$P(v_1 = 1) = .5025$

$P(v_2 = 1) = .49975$

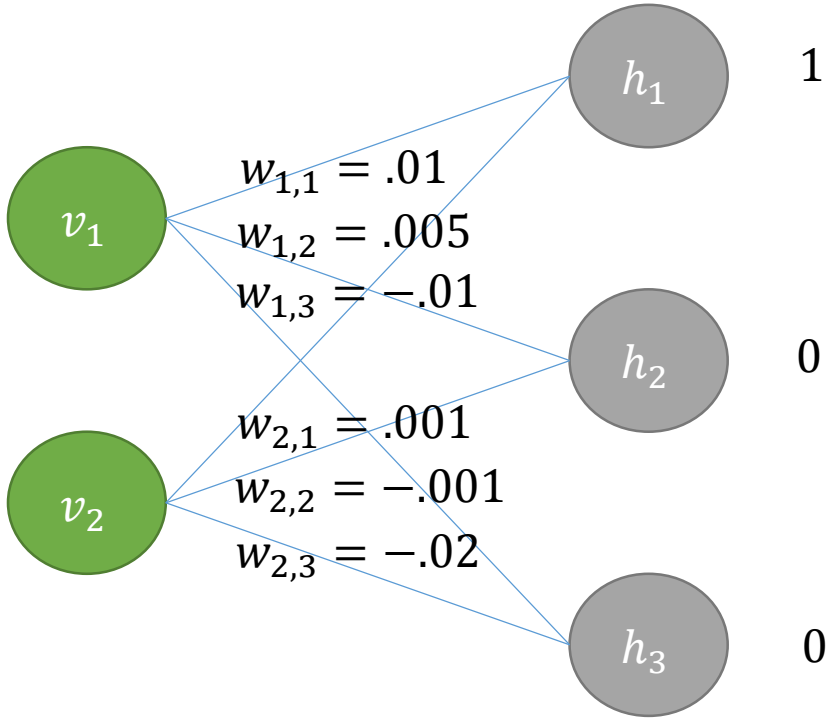


Given these probability distributions we create a random sample for the visible items.

$V_1 = <1,0>$
$H_1 = <1,0,0>$
$V_2 = ?$
$H_2 = ?$

$1 \leftsquigarrow P(v_1 = 1) = .5025$

$1 \leftsquigarrow P(v_2 = 1) = .49975$

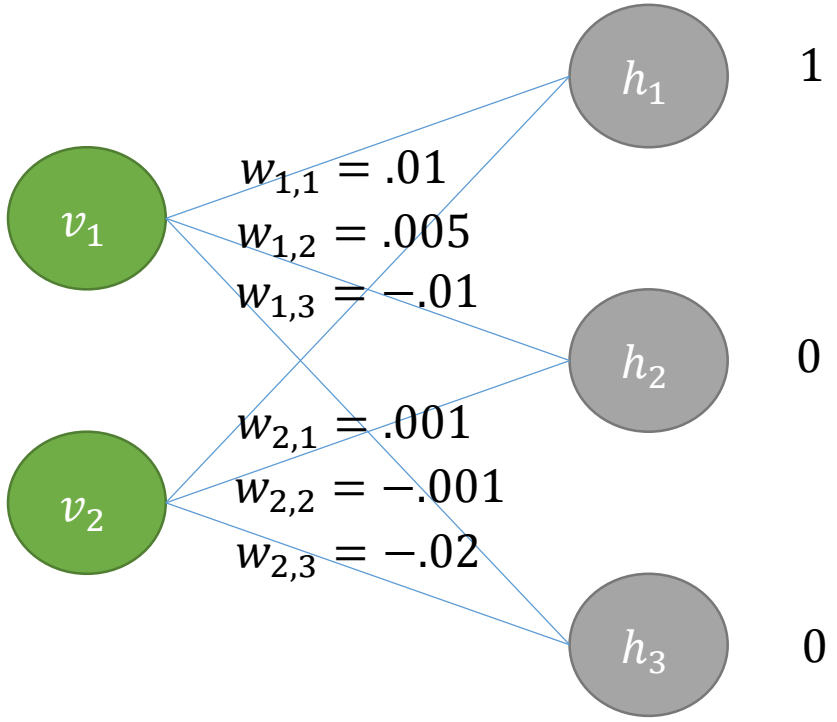


Given these probability distributions we create a random sample for the visible items.

$V_1 =$ <1,0>
$H_1 =$ <1,0,0>
$V_2 =$
$H_2 = ?$

$\mathbf{1} \leftsquigarrow P(v_1 = 1) = .5025$

$\mathbf{1} \leftsquigarrow P(v_2 = 1) = .49975$

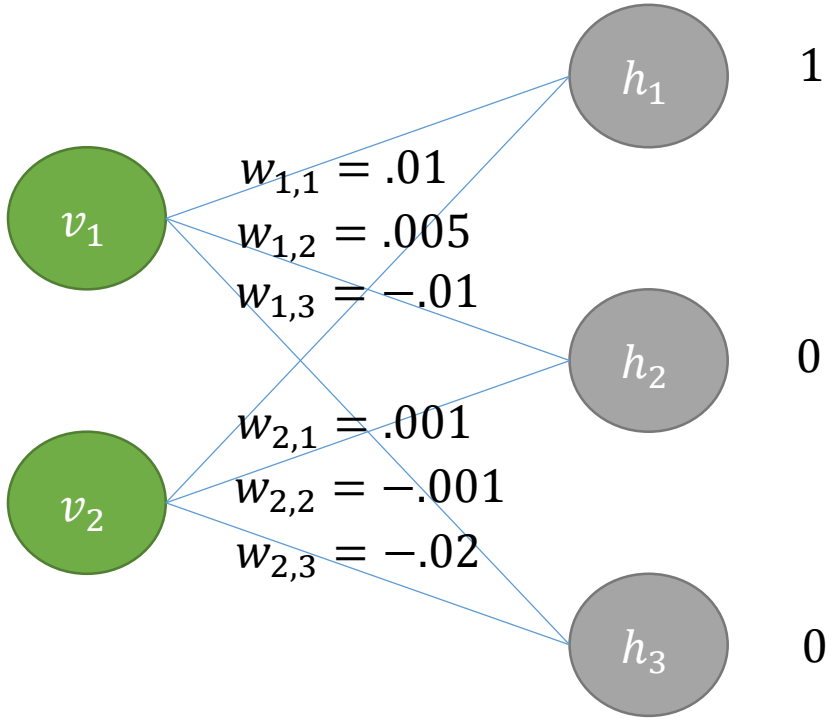


This is V_2

$V_1 = <1,0>$
$H_1 = <1,0,0>$
$\boldsymbol{V_2 = <1,1>}$
$H_2 = ?$

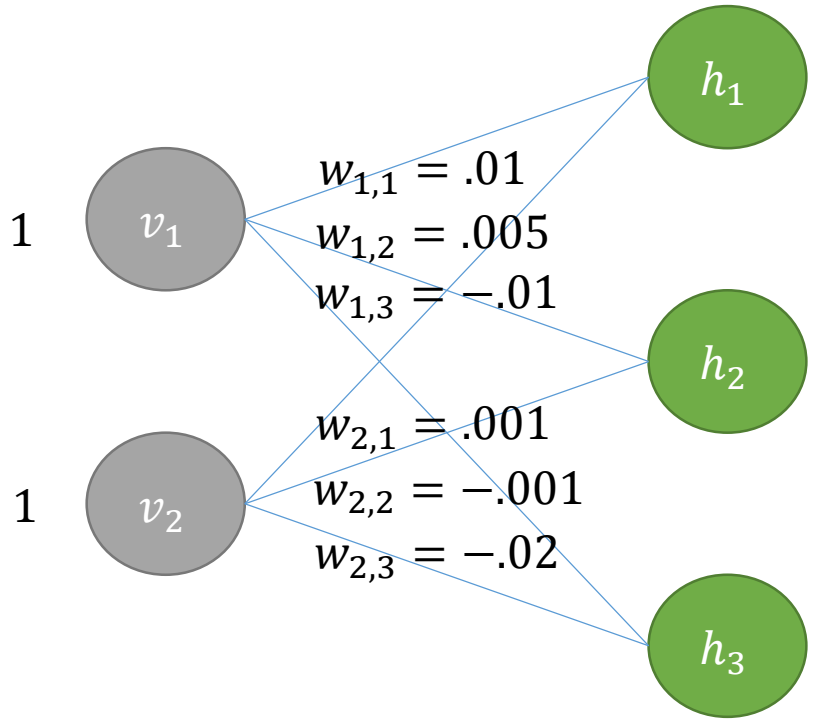
$\mathbf{1} \leftsquigarrow P(v_1 = 1) = .5025$

$\mathbf{1} \leftsquigarrow P(v_2 = 1) = .49975$



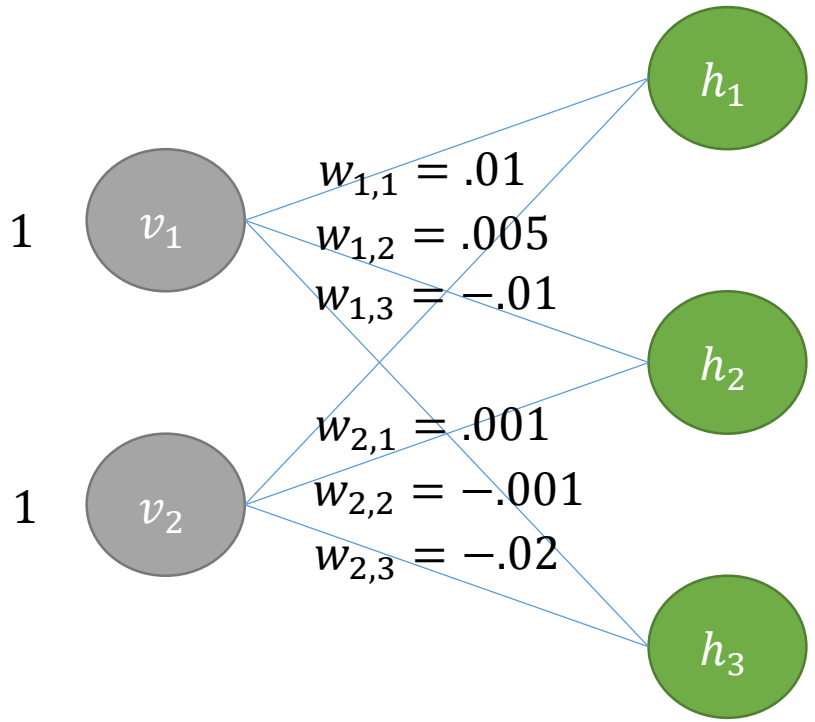
Now to find H_2 we 'unlock' the hidden variables and 'lock' the visible variables to V_2.

$V_1 =$ <1,0>
$H_1 =$ <1,0,0>
$V_2 = < 1,1 >$
$H_2 = ?$



We would do this just like before…

$V_1 = <1,0>$
$H_1 = <1,0,0>$
$V_2 = < 1,1 >$
$H_2 = ?$



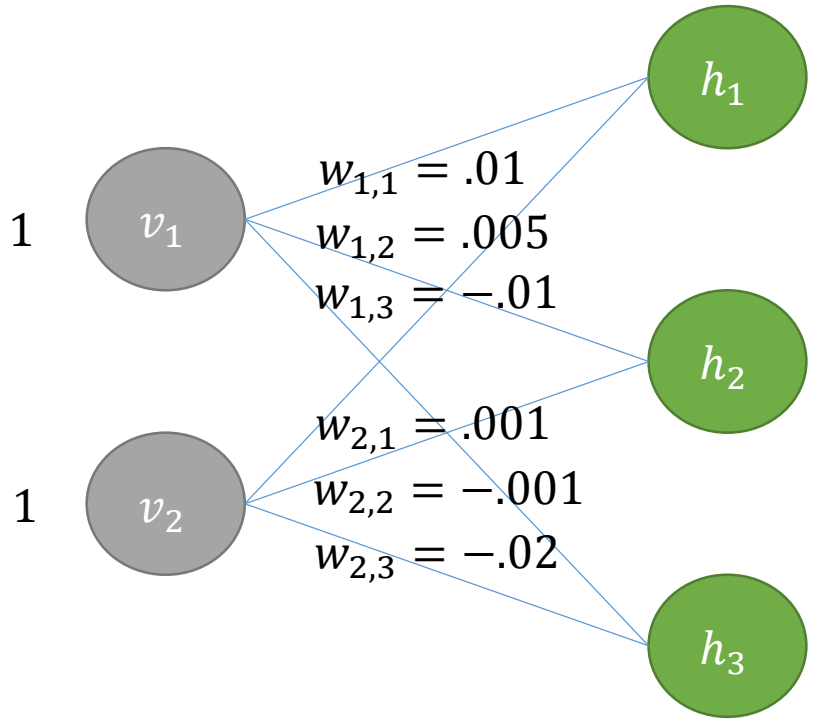
$$P(h_1 = 1) = \sigma(\sum_{j=0}^{2} v_j \, w_{j,1})$$

$$P(h_2 = 1) = \sigma(\sum_{j=0}^{2} v_j \, w_{j,2})$$

$$P(h_1 = 1) = \sigma(\sum_{j=0}^{2} v_j \, w_{j,2})$$

...and also like before we would sample from the resulting distributions.

$V_1 = <1,0>$
$H_1 = <1,0,0>$
$V_2 = < 1,1 >$
$H_2 = ?$



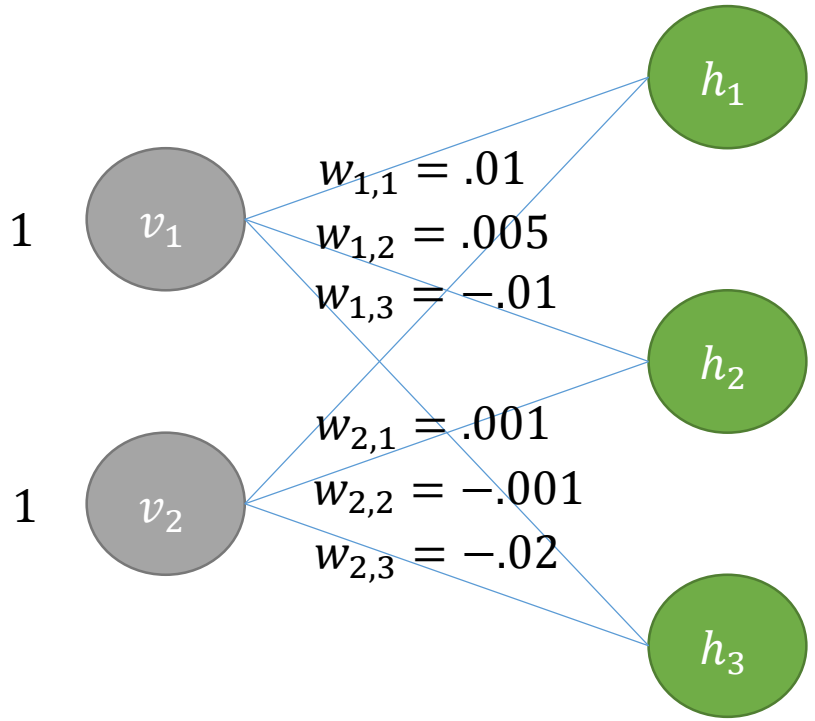
$$P(h_1 = 1) = \sigma(\sum_{j=0}^{2} v_j \, w_{j,1}) \rightsquigarrow \mathbf{1}$$

$$P(h_2 = 1) = \sigma(\sum_{j=0}^{2} v_j \, w_{j,2}) \rightsquigarrow \mathbf{1}$$

$$P(h_1 = 1) = \sigma(\textstyle\sum_{j=0}^{2} v_j \, w_{j,2}) \rightsquigarrow \mathbf{0}$$

And this would be H_2.

$V_1 = <1,0>$
$H_1 = <1,0,0>$
$V_2 = < 1,1 >$
$H_2 = <1,1,0>$



$$P(h_1 = 1) = \sigma(\sum_{j=0}^{2} v_j\, w_{j,1}) \rightsquigarrow \mathbf{1}$$

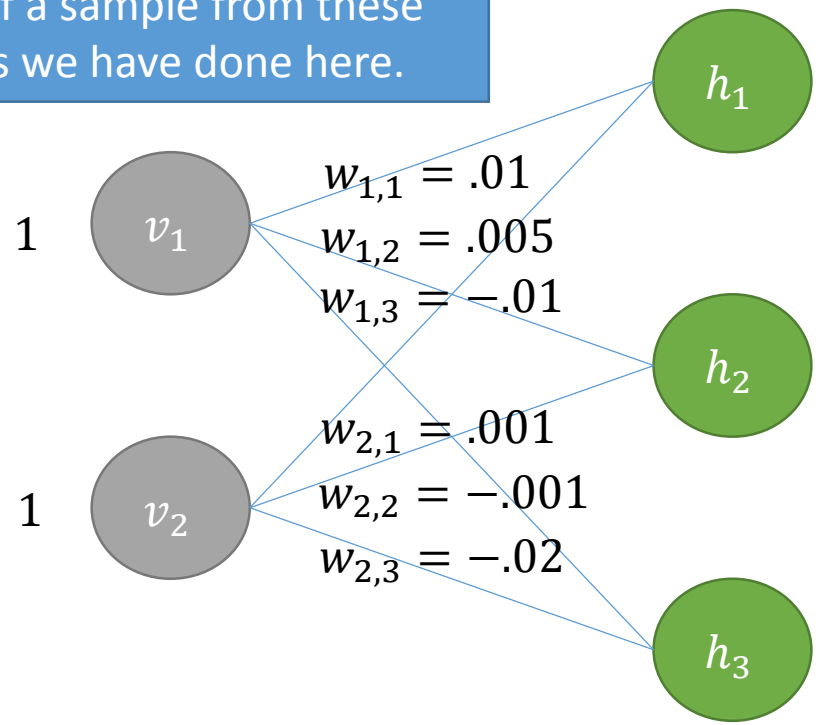
$$P(h_2 = 1) = \sigma(\sum_{j=0}^{2} v_j\, w_{j,2}) \rightsquigarrow \mathbf{1}$$

$$P(h_1 = 1) = \sigma(\textstyle\sum_{j=0}^{2} v_j\, w_{j,2}) \rightsquigarrow \mathbf{0}$$

And this would be H_2.

The most commen improvement over the naive algorithm I gave is to set H_2 to the probability of the hidden items at this point, instead of a sample from these probabilities as we have done here.

$V_1 =$ <1,0>
$H_1 =$ <1,0,0>
$V_2 =$ < 1,1 >
$H_2 =$ <1,1,0>



$$P(h_1 = 1) = \sigma(\sum_{j=0}^{2} v_j\, w_{j,1}) \rightsquigarrow \mathbf{1}$$

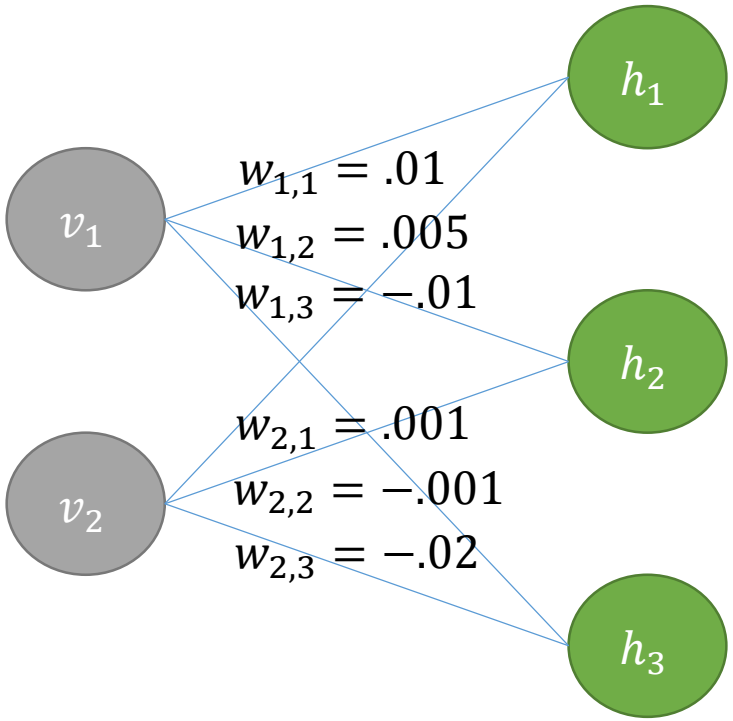
$$P(h_2 = 1) = \sigma(\sum_{j=0}^{2} v_j\, w_{j,2}) \rightsquigarrow \mathbf{1}$$

$$P(h_1 = 1) = \sigma(\textstyle\sum_{j=0}^{2} v_j\, w_{j,2}) \rightsquigarrow \mathbf{0}$$

Now we can update the weights.

$$\Delta w_{i,j} = \epsilon\left(\langle v_{1,i} h_{1,j}\rangle - \langle v_{2,i} h_{2,j}\rangle\right)$$

$V_1 =$ <1,0>
$H_1 =$ <1,0,0>
$V_2 = <\ 1,1\ >$
$H_2 =$ <1,1,0>

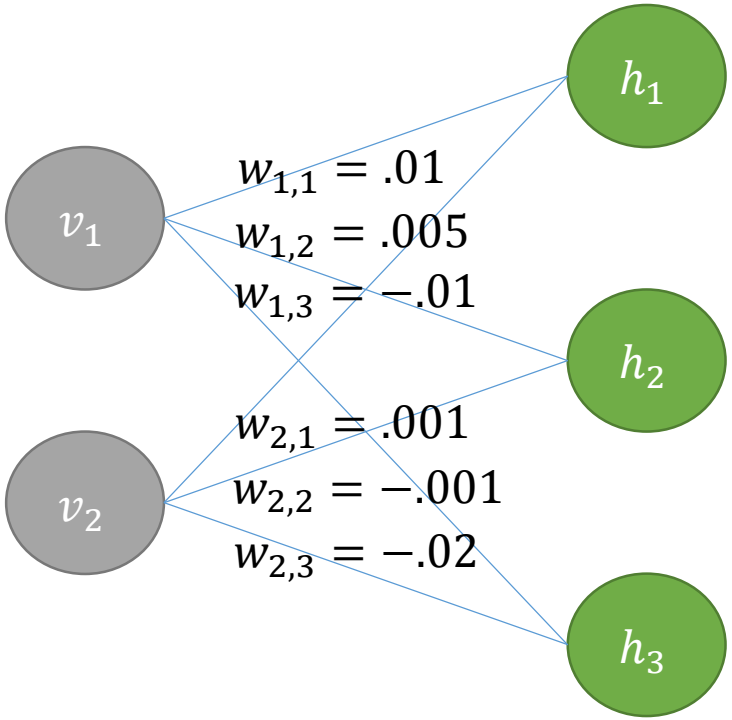


Now we can update the weights.

$$\epsilon = .01$$

$$\Delta w = \epsilon(\langle v_1 h_1 \rangle - \langle v_2 h_2 \rangle)$$

$V_1 = <1,0>$
$H_1 = <1,0,0>$
$V_2 = < 1,1 >$
$H_2 = <1,1,0>$



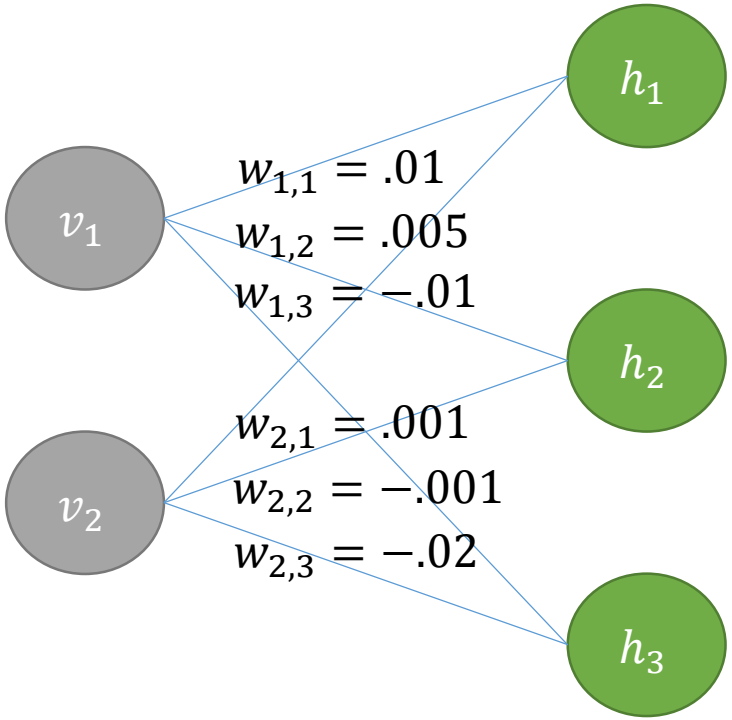
$$\Delta w = \epsilon\left(\langle v_{1,i} h_{1,j} \rangle - \langle v_{2,i} h_{2,j} \rangle\right)$$
$$= (.01)(\langle < 1,0 >< 1,0,0 > \rangle - \langle < 1,1 >< 1,1,0 > \rangle)$$
$$= (.01)\left(\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix}\right)$$
$$= (.01)\left(\begin{bmatrix} 0 & -1 & 0 \\ -1 & -1 & 0 \end{bmatrix}\right)$$
$$= \begin{bmatrix} 0 & -.01 & 0 \\ -.01 & -.01 & 0 \end{bmatrix}$$

Now we can update the weights.

$$\epsilon = .01$$

$$\Delta w = \epsilon(\langle v_1 h_1 \rangle - \langle v_2 h_2 \rangle)$$

$V_1 = <1,0>$
$H_1 = <1,0,0>$
$V_2 = < 1,1 >$
$H_2 = <1,1,0>$



So:

$$\Delta w = \begin{bmatrix} 0 & -.01 & 0 \\ -.01 & -.01 & 0 \end{bmatrix}$$

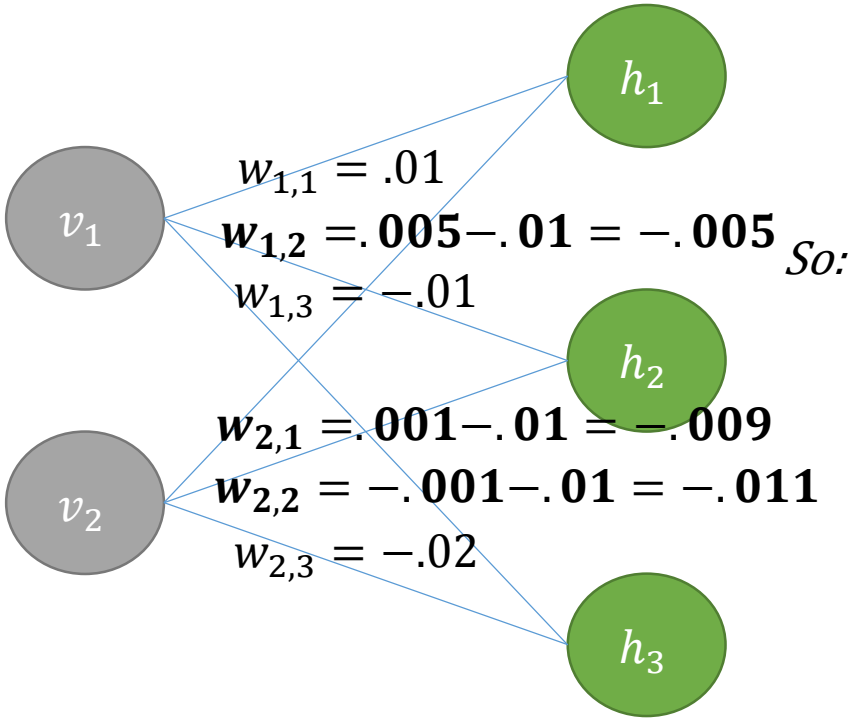
$\Delta w_{1,1} = 0$
$\Delta w_{1,2} = -.01$
$\Delta w_{1,3} = 0$
$\Delta w_{2,1} = -.01$
$\Delta w_{2,2} = -.01$
$\Delta w_{2,3} = 0$

Now we can update the weights.

$$\epsilon = .01$$

$$\Delta w = \epsilon(\langle v_1 h_1 \rangle - \langle v_2 h_2 \rangle)$$

$V_1 =$ <1,0>
$H_1 =$ <1,0,0>
$V_2 =$ < 1,1 >
$H_2 =$ <1,1,0>

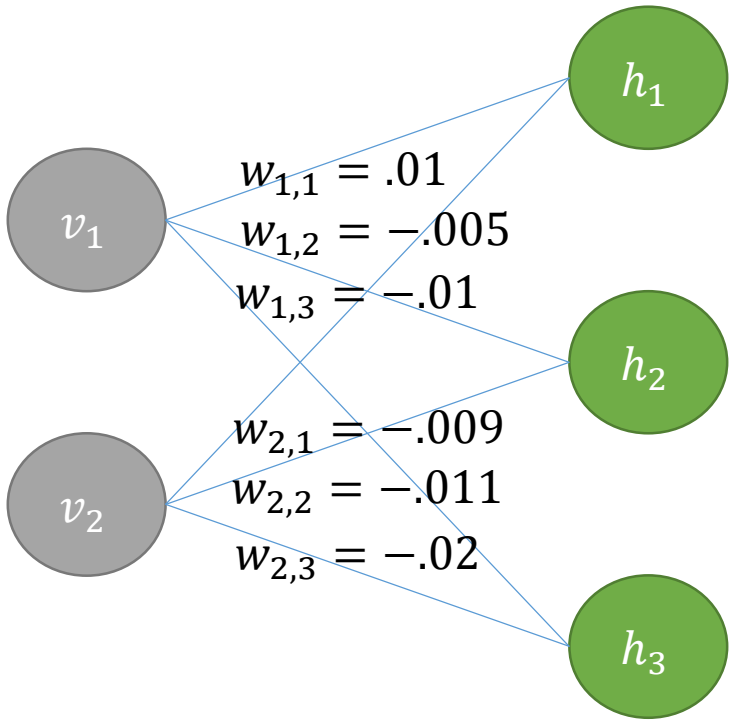


So:

$$\Delta w = \begin{bmatrix} 0 & -.01 & 0 \\ -.01 & -.01 & 0 \end{bmatrix}$$

$\Delta w_{1,1} = 0$
$\Delta w_{1,2} = -.01$
$\Delta w_{1,3} = 0$
$\Delta w_{2,1} = -.01$
$\Delta w_{2,2} = -.01$
$\Delta w_{2,3} = 0$

Now we do that for all items in the dataset. And we repeat going through the whole dataset until convergence…



Contrastive Divergence

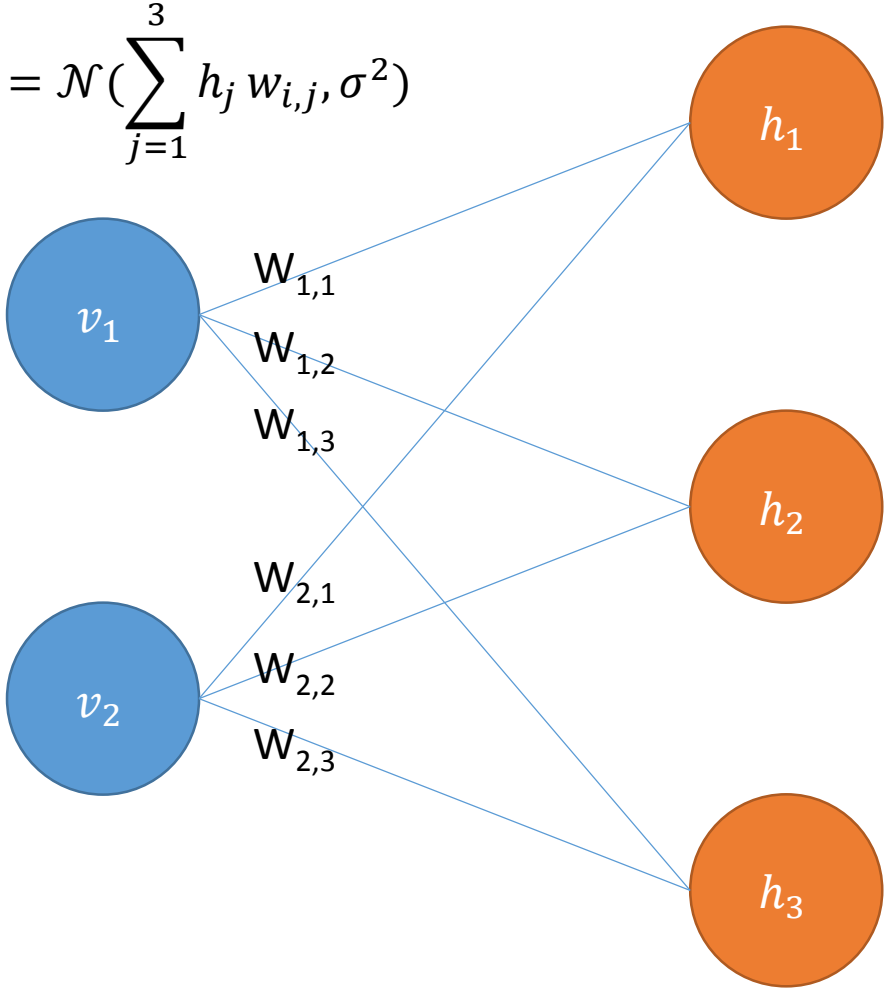
- The algorithm is stochastic (random): It is only probable that it will take us in the right direction each time.
  - Taking the probabilities for H_2 improves things a lot.
  - Increasing *n* improves things a lot – at a big time cost!
- Only by doing this lots of times can we expect it to actually move in the right direction with high probability.

(Real) Restricted Boltzmann Machines

- Visible units:
  - Normally distributed
  - If scaled and centered, σ^2=1
  - Note σ has different meaning in each equation!

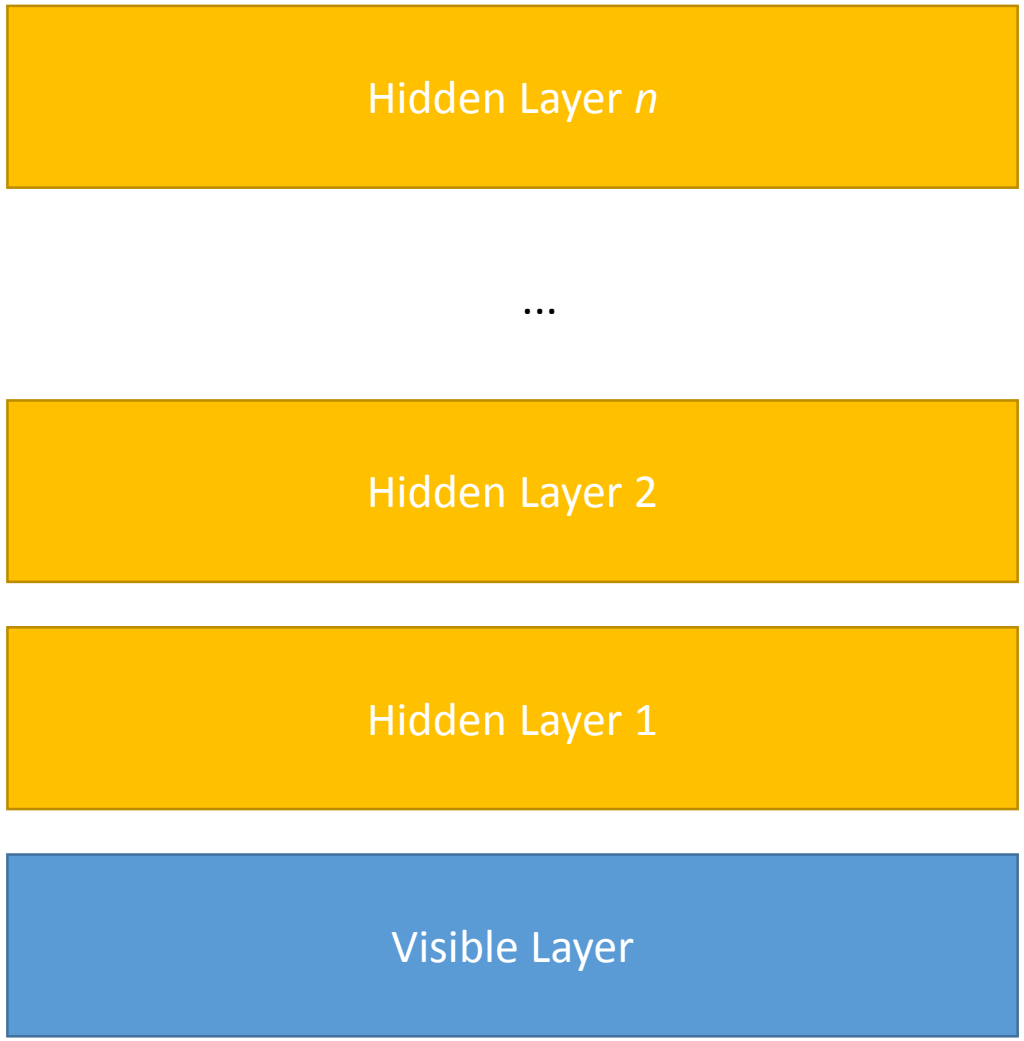
$$P(H_i = 1) = \sigma(\sum_{j=0}^{2} v_j \, w_{j,i})$$

$$P(v_i) = \mathcal{N}(\sum_{j=1}^{3} h_j \, w_{i,j}, \sigma^2)$$



Deep Belief Networks

- Stacked RBMs
- The first hidden layer is trained as usual.
- Then, each higher layer, *i*, is trained by:
  - Generating training inputs for layer *i*-1 from the data and any intermediate layers.
  - Using these training inputs to train layers *i*-1 and *i* as a usual RBM.



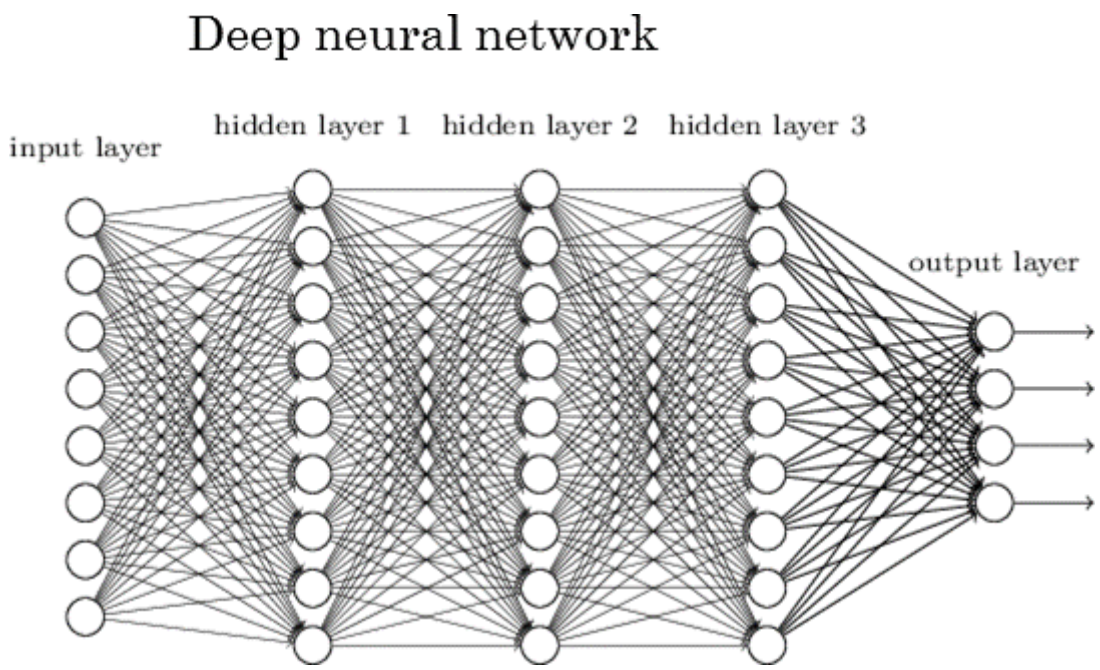
DBNs and DNNs

Gradient descent training algorithms (like back-prop) for neural networks do not perform well on deep networks.

- Gradients disappear
- Too many local minima

Instead ’layers trained separately’.

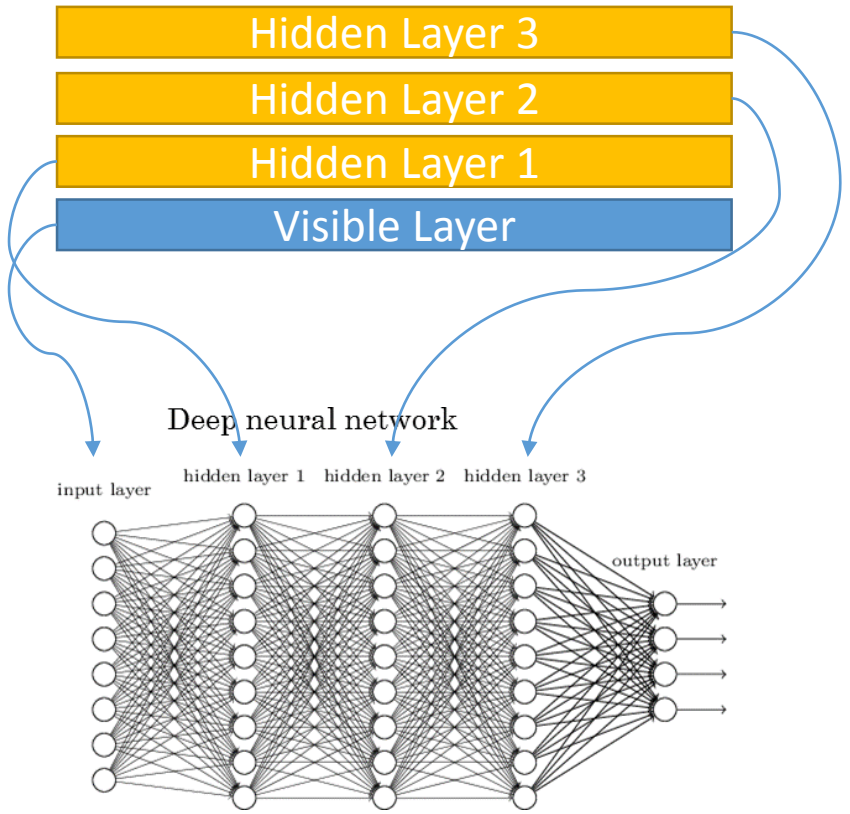
DBNs one method of doing so.



DBNs and DNNs

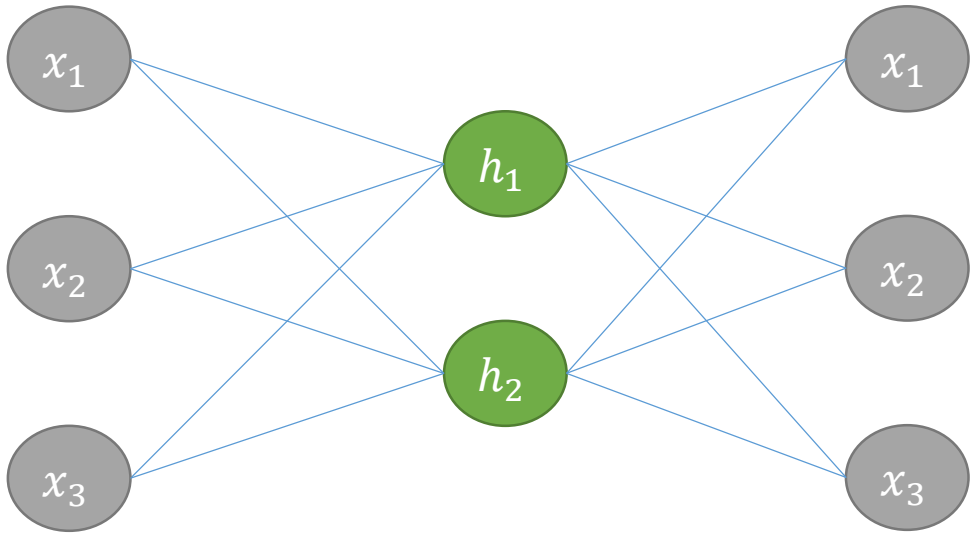
Process of using DBNs as pre-training for DNNs:

1. Train a DBN.
2. Use the parameters from the DBN as the initial parameters for DNN (up to last hidden layer).
3. Do fine tuning with a gradient descent algorithm.



Auto-Encoders

- Neural Networks that seek to find alternative encoding of the data.
- Outputs are the input vectors.
- Trained as any (shallow) neural network.

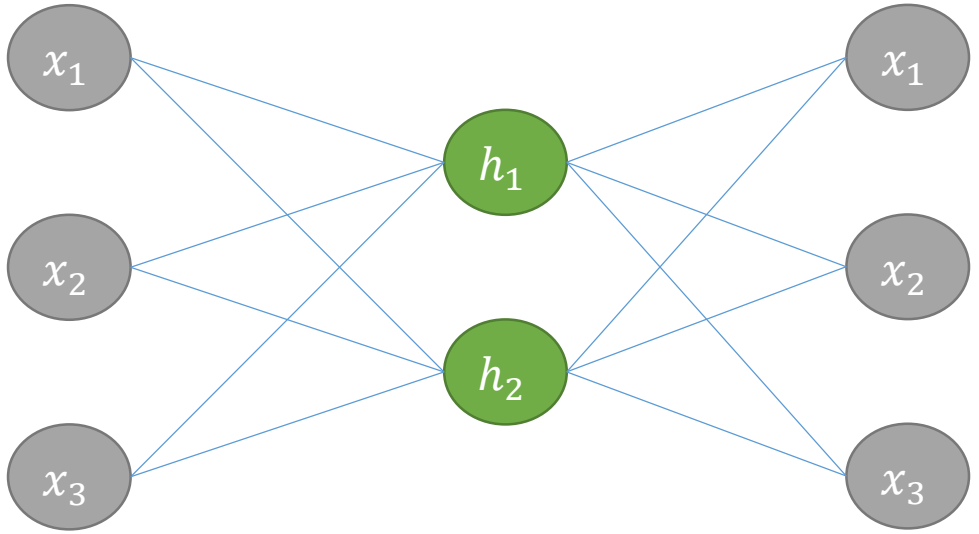


Auto-Encoders

Need to avoid the network learning a trivial identity function.

This can be done by:

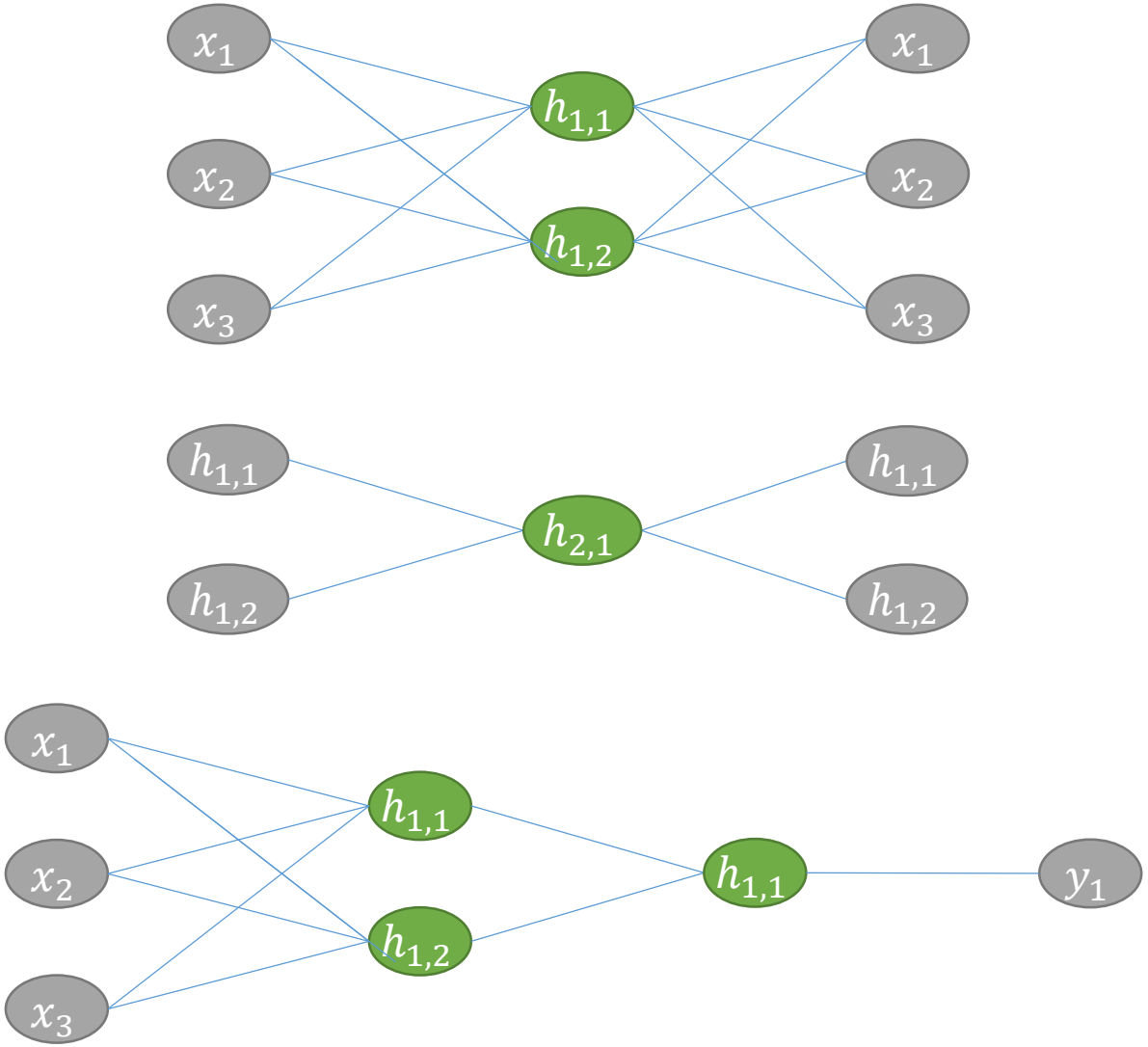
- Having fewer nodes in the hidden layer than in the input/output layers.
- Using regularisation techniques. For example:
  - restricting the values the weights can take.
  - adding noise to the training data



Auto-Encoders & DNNs

Auto-encoders can be stacked just like RBNs to obtain another method of pre-training DNNs.

- The first hidden layer is trained as an autoencoder from the input data as usual.
- Then, each higher layer, *i*, is trained by:
  - Generating training inputs for layer *i*-1 from the input data and any intermediate layers.
  - Using these training inputs to train layers *i*-1 and *i* as a usual autoencoder.
- Use the parameters from these autoencoders as the initial parameters for DNN (up to last hidden layer).
- Do fine tuning with a gradient descent algorithm.



Why deep learning?

- Impressive performance in many areas.
- Hope for possibility of transfer learning.
  - Perhaps different but related domains will share higher parts of the network but have individual lower parts. For example, if the input variables are linguistic tokens and the output variable is meaning you might hope that a deep network would permit multiple languages to share the deep part of the network.
    - The shallow parts of the network would be language dependent and transform the linguistic tokens of an actual language into more abstract linguistic features.
    - The deep part of the network would use these shared abstract linguistic features to inpute the meaning.
  - My opinion: Interesting... but let’s wait for clear evidence that it can be done!