

# **Basic Deep Learning Models**

---

*Dong Yu*

*Microsoft Research*

# Part 2: Outline

---

- **Energy-based model (EBM)**
- Restricted Boltzmann machine (RBM)
- Deep belief network (DBN)
- Higher-order Boltzmann machines (HOBM)
- Recurrent neural network (RNN)

# Energy Based Model

- Each configuration is associated with an energy function  $E(x)$ 
  - $x$  can contain feature  $v$ , label  $y$  and hidden state  $h$
- Energy and probability is convertible
  - If energy is known:  $p(x) = \frac{e^{-\beta E(x)}}{\int_{x'} e^{-\beta E(x')} dx'}$
  - If probability is known:  $E(x) \propto -\log p(x)$
- Execution: move the system to the stable low energy configuration
  - Deterministically: normal stable state
  - Stochastically: thermal stable state (equilibrium)
- Training: adjust model parameters to
  - Give high probability (or low energy) to good answers
  - Give low probability (or high energy) to bad answers

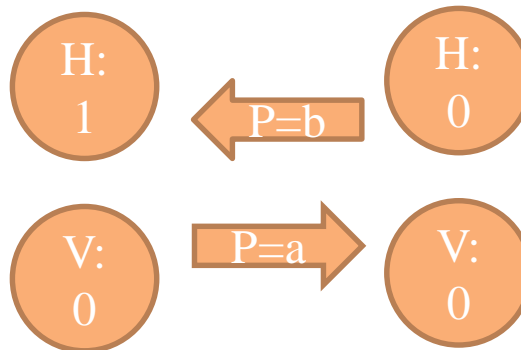
# Energy Based Model: Execution

$$E(x) = (h - v)^2$$

Deterministically



Stochastically

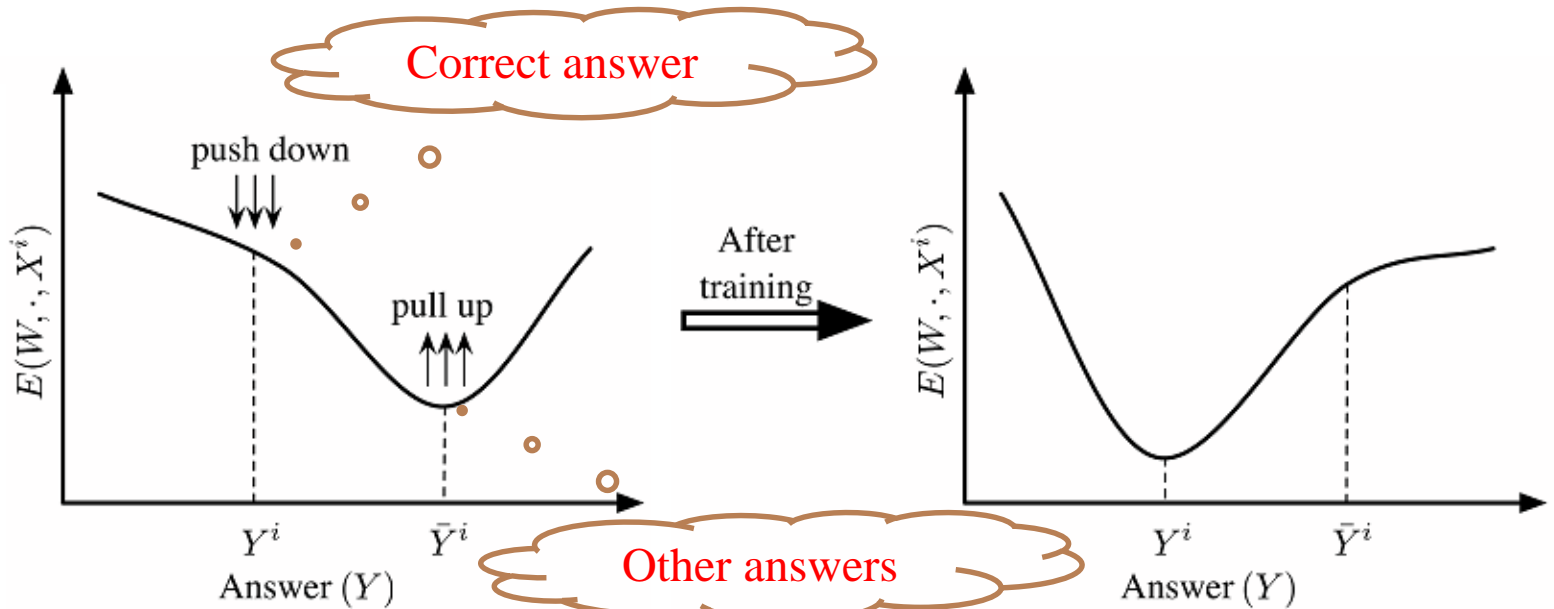
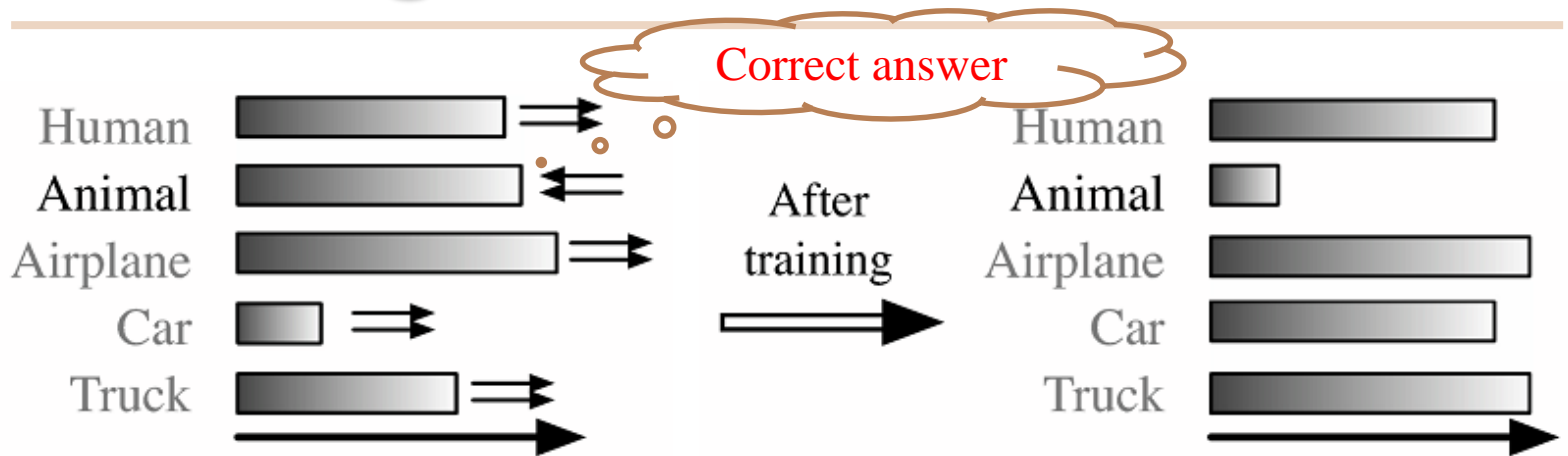


# Loss Function

- Is defined on the energy function over the whole training set (but different from the energy function itself)
- Designed to solve the problem in hand
- **Classification**: which label  $y$  is most compatible with feature  $v$ ?
  - loss function: when minimized energy  $E(v, y)$  is lowest with the correct answer
- **Ranking**: Is  $y_1$  or  $y_2$  more compatible with  $v$ ?
  - loss function: when minimized energy  $E(v, y)$  ranks the answers correctly
- **Detection**: Is the value  $y$  compatible with  $v$ ?
  - loss function: when minimized energy  $E(v, y)$  increases as  $y$  incompatible with  $v$ .
- **Conditional density estimation**: What is the conditional distribution  $p(y|v)$ ?
  - loss function: maximize conditional probability defined on energy  $E(v, y)$

# Training

EBM | RBM | DBN | HOBM | RNN



Slide credit Yann Lecun

# Free Energy

- When hidden variable  $h$  is needed  $x = (v, h)$ , e.g., if
  - the example is not fully observable, or
  - to increase the expressive power of the model
- Introduce free energy **Visible variables clamped**

$$F(v) = -\log \int_h e^{-E(v,h)} dh$$

- Benefits
  - Probability has same form as that without hidden variable:

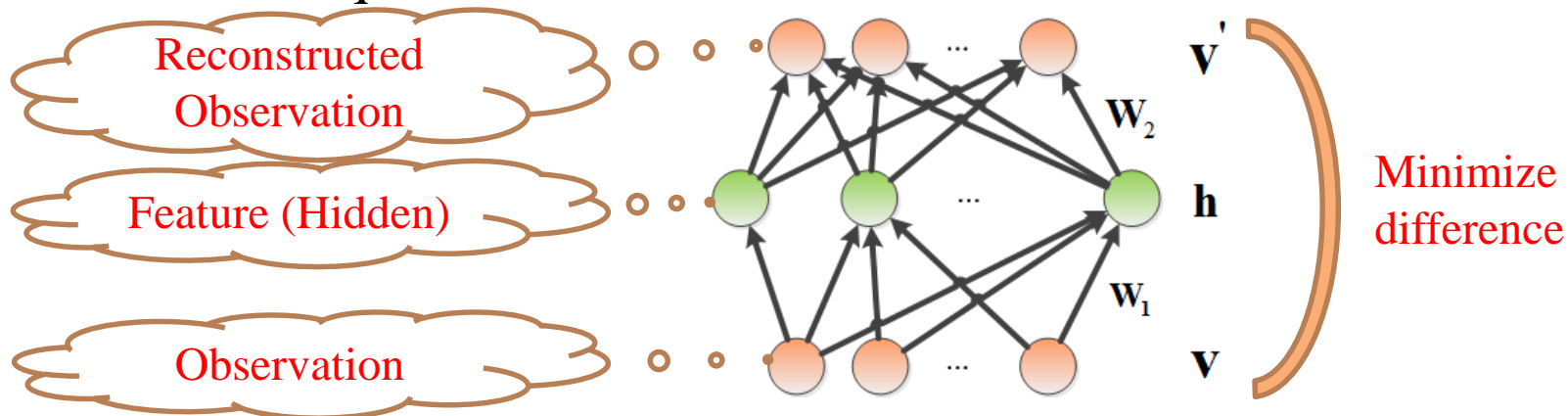
$$p(v) = \frac{e^{-\beta F(v)}}{\int_{v'} e^{-\beta F(v')} dv'}$$

- Gradient has a simple form:

$$-\frac{\partial \log p(v)}{\partial \theta} = \frac{\partial F(v)}{\partial \theta} - \int_{\tilde{v}} p(\tilde{v}) \frac{\partial F(\tilde{v})}{\partial \theta} d\tilde{v}$$

# Unsupervised Feature Extraction

- What is good feature?
  - The answer is task dependent ☹️
  - Basic attributes: invariant and selective
  - Other attributes: easy and efficient to extract
- Feature extraction with auto-encoder
  - The feature should conserve information to reconstruct the original signal
  - Interesting since it conserves major information and do not require labeled data



# Energy based Auto-Encoder

- Total energy:

$$E(v, h, \mathbf{W}_c, \mathbf{W}_d) = \alpha E_c(v, h, \mathbf{W}_c) + (1 - \alpha) E_d(v, h, \mathbf{W}_d)$$

- Encoding energy:

$$E_c(v, h, \mathbf{W}_c) = \|h - f(\mathbf{W}_c v)\|^2$$

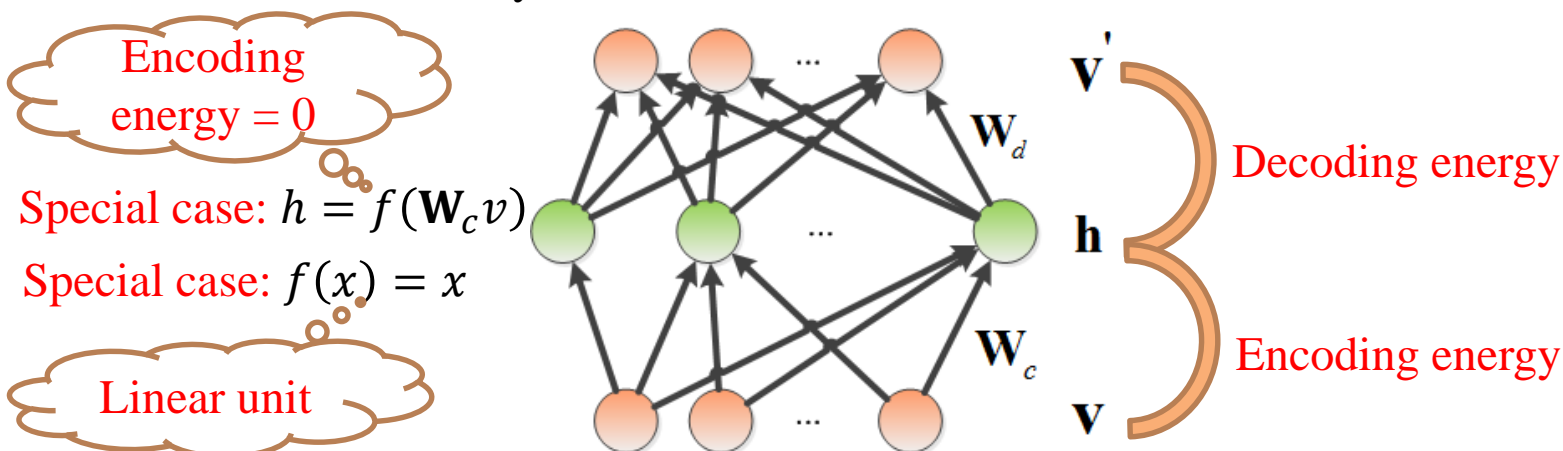
- Decoding energy:

$$E_d(v, h, \mathbf{W}_d) = \|v - g(\mathbf{W}_d h)\|^2$$

- Conditional probability:

$$p(h|v, \mathbf{W}_c, \mathbf{W}_d) \propto e^{-\beta E(v, h, \mathbf{W}_c, \mathbf{W}_d)}$$

- Reduce dimensions
- Increase dimensions: trivial solutions exist (use noisy input) -> small variation defined by the noise does not affect the feature.

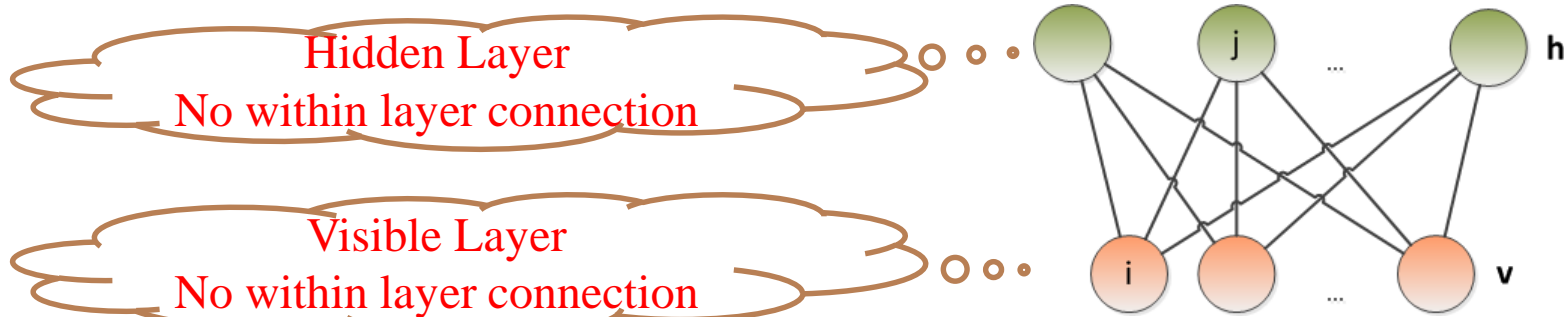


# Part 2: Outline

---

- Energy-based model (EBM)
- **Restricted Boltzmann machine (RBM)**
- Deep belief network (DBN)
- Higher-order Boltzmann machines (HOBM)
- Recurrent neural network (RNN)

# Restricted Boltzmann Machine



- Joint distribution  $p(\mathbf{v}, \mathbf{h}; \theta)$  is defined in terms of an energy function  $E(\mathbf{v}, \mathbf{h}; \theta)$

Is an energy based model. Runs stochastically

$$p(\mathbf{v}, \mathbf{h}; \theta) = \frac{\exp(-E(\mathbf{v}, \mathbf{h}; \theta))}{Z}$$

Free energy

$$p(\mathbf{v}; \theta) = \sum_{\mathbf{h}} \frac{\exp(-E(\mathbf{v}, \mathbf{h}; \theta))}{Z} = \frac{\exp(-F(\mathbf{v}; \theta))}{Z}$$

- Conditional independence

$$p(\mathbf{h}|\mathbf{v}) = \prod_{j=0}^{H-1} p(h_j|\mathbf{v})$$

$$p(\mathbf{v}|\mathbf{h}) = \prod_{i=0}^{V-1} p(v_i|\mathbf{h})$$

# Energies in RBM

One explanation on the meaning of the energy

Maximum entropy with constraints on each neuron and neuron pair's average firing rate or prediction error.

- For a Bernoulli-Bernoulli RBM

$$E(\mathbf{v}, \mathbf{h}; \theta) = - \sum_{i=1}^V \sum_{j=1}^H w_{ij} v_i h_j - \sum_{i=1}^V b_i v_i - \sum_{j=1}^H a_j h_j$$

Lagrange multiplier for the pair

Lagrange multiplier for each neuron

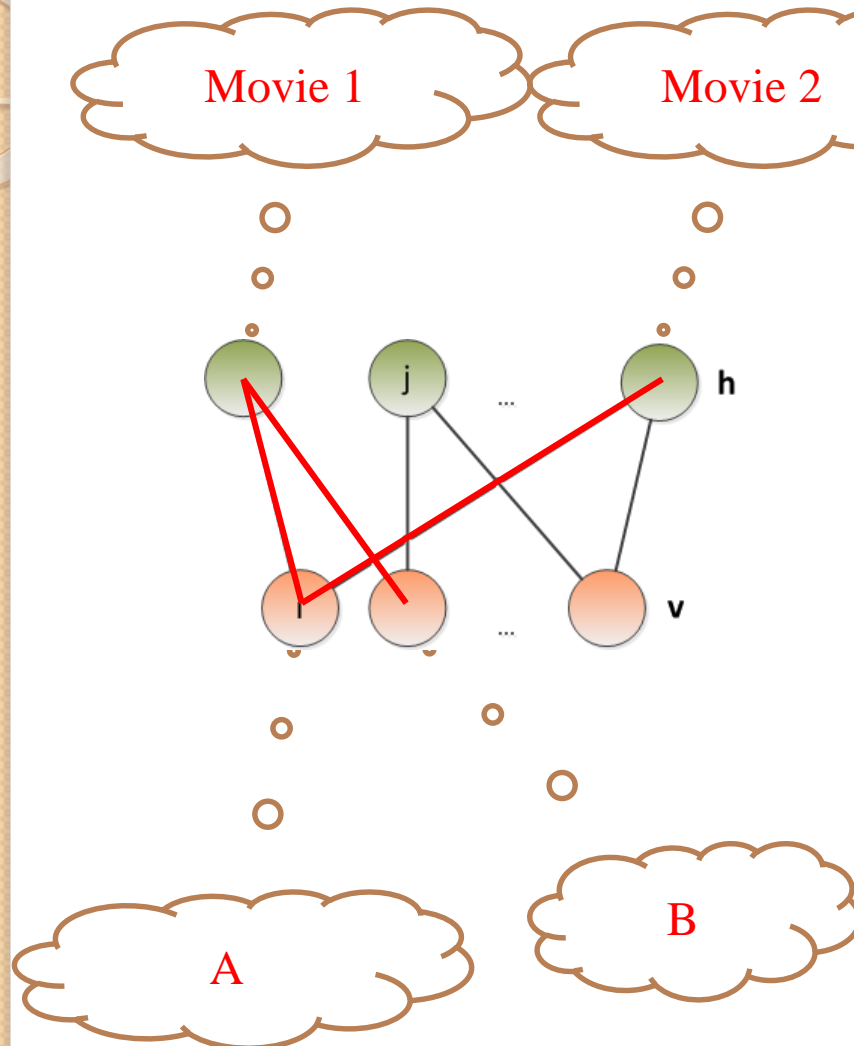
- For a Gaussian-Bernoulli RBM

$$E(\mathbf{v}, \mathbf{h}; \theta) = - \sum_{i=1}^V \sum_{j=1}^H w_{ij} v_i h_j + \frac{1}{2} \sum_{i=1}^V (v_i - b_i)^2 - \sum_{j=1}^H a_j h_j$$

Lagrange multiplier for the pair

Minimize prediction error

# RBM Can Model Correlations



- All people who like movie 1
- All movies A likes

# Restricted Boltzmann Machine

- **Good news:** Conditional probabilities are very easy to calculate
- For a Bernoulli-Bernoulli RBM

Inference  $\dots p(h_j = 1 | \mathbf{v}; \theta) = \sigma \left( \sum_{i=1}^V w_{ij} v_i + a_j \right)$

Generation  $\dots p(v_i = 1 | \mathbf{h}; \theta) = \sigma \left( \sum_{j=1}^H w_{ij} h_j + b_i \right)$

- For a Gaussian-Bernoulli RBM

Inference  $\dots p(h_j = 1 | \mathbf{v}; \theta) = \sigma \left( \sum_{i=1}^V w_{ij} v_i + a_j \right)$

Generation  $\dots p(v_i | \mathbf{h}; \theta) = N \left( \sum_{j=1}^H w_{ij} h_j + b_i, 1 \right)$

The prediction error (not observation) is Gaussian

# Learning RBM Parameters

probabilities to observe visible vectors if we take random samples after the whole network has reached thermal equilibrium

Expected value of product of states at thermal equilibrium when the training vector is clamped on the visible units

$$\frac{\partial \log p(\mathbf{v})}{\partial w_{ij}} = \langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model}$$

## Intuition

When well trained the model predicted joint expectation matches that observed in the data

Derivative of log probability of one training vector

Expected value of product of states at thermal equilibrium when nothing is clamped

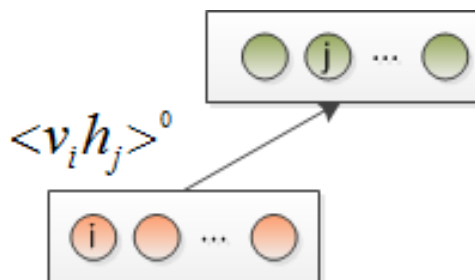
**Intractable!!!**

# Learning RBM Parameters



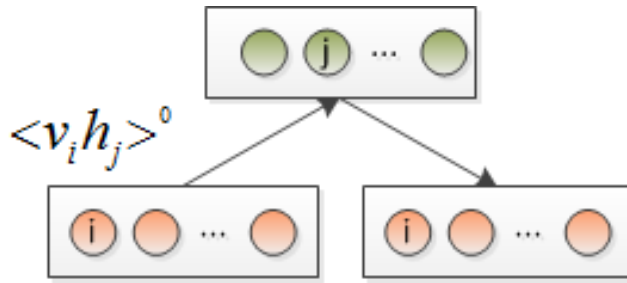
- $\frac{\partial \log p(\mathbf{v})}{\partial w_{ij}} = \langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model}$
- Approximate  $\langle v_i h_j \rangle_{model}$  with Gibb's sampling
  - i. Initialize  $\mathbf{v}_0$  at data
  - ii.
  - iii.
  - iv.

# Learning RBM Parameters



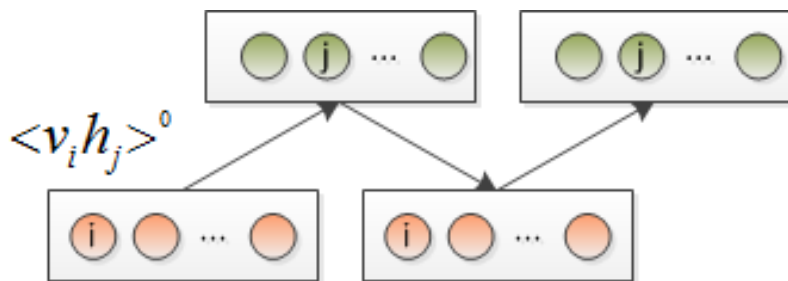
- $\frac{\partial \log p(\mathbf{v})}{\partial w_{ij}} = \langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model}$
- Approximate  $\langle v_i h_j \rangle_{model}$  with Gibb's sampling
  - i. Initialize  $\mathbf{v}_0$  at data
  - ii. Sample  $\mathbf{h}_0 \sim p(\mathbf{h}|\mathbf{v}_0)$
  - iii.
  - iv.

# Learning RBM Parameters



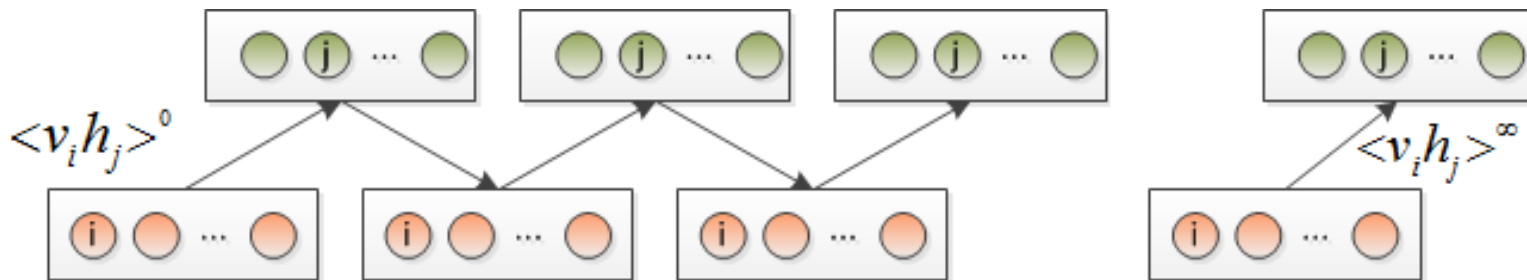
- $\frac{\partial \log p(\mathbf{v})}{\partial w_{ij}} = \langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model}$
- Approximate  $\langle v_i h_j \rangle_{model}$  with Gibb's sampling
  - i. Initialize  $\mathbf{v}_0$  at data
  - ii. Sample  $\mathbf{h}_0 \sim p(\mathbf{h}|\mathbf{v}_0)$
  - iii. Sample  $\mathbf{v}_1 \sim p(\mathbf{v}|\mathbf{h}_0)$
  - iv.

# Learning RBM Parameters



- $\frac{\partial \log p(\mathbf{v})}{\partial w_{ij}} = \langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model}$
- Approximate  $\langle v_i h_j \rangle_{model}$  with Gibb's sampling
  - i. Initialize  $\mathbf{v}_0$  at data
  - ii. Sample  $\mathbf{h}_0 \sim p(\mathbf{h}|\mathbf{v}_0)$
  - iii. Sample  $\mathbf{v}_1 \sim p(\mathbf{v}|\mathbf{h}_0)$
  - iv. Sample  $\mathbf{h}_1 \sim p(\mathbf{h}|\mathbf{v}_1)$

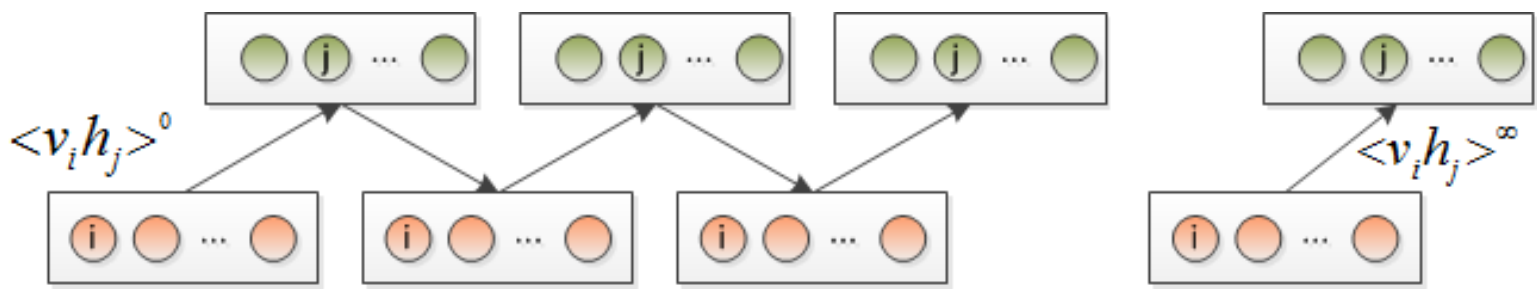
# Learning RBM Parameters



- $(\mathbf{v}_\infty, \mathbf{h}_\infty)$  is a true sample from the model.  $(\mathbf{v}_1, \mathbf{h}_1)$  is a very rough estimate
- Contrastive divergence algorithm (CD)
- $$\frac{\partial \log p(\mathbf{v})}{\partial w_{ij}} = \langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model} \cong \langle v_i h_j \rangle_0 - \langle v_i h_j \rangle_1$$
- Approximate  $\langle v_i h_j \rangle_{model}$ 
  - i. Initialize  $\mathbf{v}_0$  at data
  - ii. Sample  $\mathbf{h}_0 \sim p(\mathbf{h}|\mathbf{v}_0)$
  - iii. Sample  $\mathbf{v}_1 \sim p(\mathbf{v}|\mathbf{h}_0)$
  - iv. Sample  $\mathbf{h}_1 \sim p(\mathbf{h}|\mathbf{v}_1)$
  - v. Call  $(\mathbf{v}_1, \mathbf{h}_1)$  a sample from the model.

Minimize reconstruction error.

# Learning RBM Parameters



- $(\mathbf{v}_\infty, \mathbf{h}_\infty)$  is a true sample from the model.  $(\mathbf{v}_1, \mathbf{h}_1)$  is a very rough estimate

- Contrastive divergence algorithm (CD)

Minimize reconstruction error.

- $$\frac{\partial \log p(\mathbf{v})}{\partial w_{ij}} = \langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model} \cong \langle v_i h_j \rangle_0 - \langle v_i h_j \rangle_1$$

- Approximate  $\langle v_i h_j \rangle_{model}$ 
  - i. Initialize  $\mathbf{v}_0$  at data
  - ii. Sample  $\mathbf{h}_0 \sim p(\mathbf{h}|\mathbf{v}_0)$
  - iii. Sample  $\mathbf{v}_1 \sim p(\mathbf{v}|\mathbf{h}_0)$
  - iv. Sample  $\mathbf{h}_1 \sim p(\mathbf{h}|\mathbf{v}_1)$
  - v. Call  $(\mathbf{v}_1, \mathbf{h}_1)$  a sample from the model.

Intuition

- $(\mathbf{v}_1, \mathbf{h}_1)$  is very noisy
- Sample  $\mathbf{h}_0$  but use  $p(\mathbf{v}|\mathbf{h}_0)$  as  $\mathbf{v}_1$  and  $p(\mathbf{h}|\mathbf{v}_1)$  as  $\mathbf{h}_1$
- Similar to noisy auto-encoder

This is not following the gradient of the log likelihood. But it works very well.

# Tracking Progress

- Because of the partition function we cannot estimate the log-likelihood during training.
- Inspection of Negative Samples
  - As training progresses, negative samples  $v_1$  should look like samples from the training set.
  - Obvious bad hyper-parameters can be filtered out
- Visual Inspection of Filters
  - Plot the weights of each unit as a gray-scale image (after reshaping to a square matrix).
  - Filters should pick out strong features in the data.
- Proxies to Likelihood
  - Cross-entropy or difference between the input and the reconstruction

# Learning RBM Parameters

- Contrastive divergence requires predetermined number of hidden units.
- Contrastive divergence is difficult to parallelize.
- Alternative learning algorithm: projection pursuit:
  - Choose initial marginal distribution (Gaussian)  $p(\mathbf{v})$
  - Find a direction along which some projection index is maximized (using EM). This is the new hidden unit and weight
  - Remove the effect of the new direction to the training samples (similar to boosting)
  - The new marginal distribution  $p(\mathbf{v})$  is adjusted.
  - Repeat
- Properties of projection pursuit
  - A greedy algorithm that adds hidden units one at a time
  - Can determine the number of hidden units needed
  - Can be parallelized across samples (using EM)
  - Work well for Gaussian-Bernoulli RBM. Tricks needed for Bernoulli-Bernoulli RBM.

# Discriminative RBM

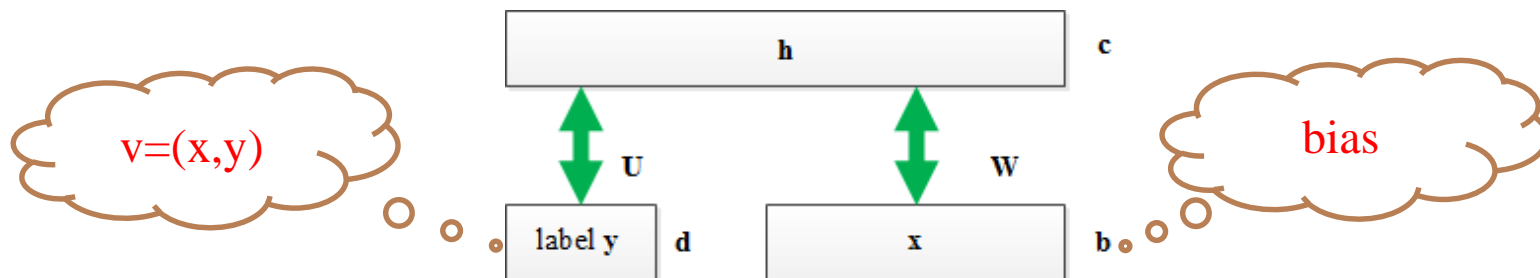
- In the generative setting we train RBM to optimize  $\log p(\mathbf{v}) = \log p(\mathbf{x}, \mathbf{y})$
- We can train the same generative model using discriminative criterion  $\log p(\mathbf{y}|\mathbf{x})$  if label  $\mathbf{y}$  is available.
- Can be trained using gradient descent or contrastive divergence algorithm.

$$\frac{\partial \log p(y = l|x)}{\partial \theta} = \sum_j \sigma(o_{lj}(x)) \frac{\partial o_{lj}(x)}{\partial \theta} - \sum_{j,y} \sigma(o_{yj}(x)) p(y|x) \frac{\partial o_{yj}(x)}{\partial \theta}$$

$$o_{yj}(x) = c_j + \sum_k W_{jk} x_k + U_{jy}$$

- When training set contains both sets with and without labels, can use weighted criterion – support semi-supervised training

$$L = \sum_{i \in \text{labeled}} \log p(y^{(i)}|x^{(i)}) + \alpha \sum_{i \in \text{labeled}} \log p(x^{(i)}, y^{(i)}) + \beta \sum_{j \in \text{unlabeled}} \log p(x^{(j)})$$



# RBM and GMM

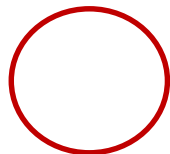
- Both are generative models
- GMM: Mixture model
  - Mixture models never produce distributions sharper than the individual components
  - Many **highly localized** Gaussians are needed to model the distribution well
  - Lots of data needed to estimate parameters
- RBM: Equivalent to mixture of product of Gaussians

$$p(\mathbf{v}) \propto e^{-\frac{1}{2}(\mathbf{v}-\mathbf{b})^T(\mathbf{v}-\mathbf{b})} \prod_j \left(1 + e^{c_j + \mathbf{v}^T \mathbf{w}_{*,j}}\right)$$

- Product models generate distributions sharper than the individual components
- Can be trained well with less data

# RBM Marginal Distribution

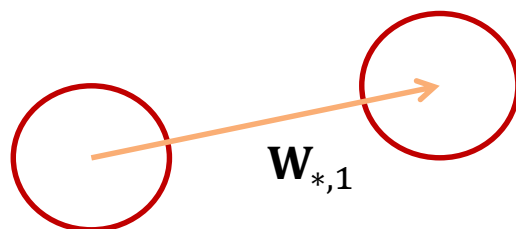
$$\begin{aligned}
 p_n(\mathbf{v}) &\propto e^{-\frac{1}{2}(\mathbf{v}-\mathbf{b})^T(\mathbf{v}-\mathbf{b})} \prod_j \left( 1 + e^{c_j + \mathbf{v}^T \mathbf{W}_{*,j}} \right) \\
 &= p_{n-1}(\mathbf{v}) \left( 1 + e^{c_n + \mathbf{v}^T \mathbf{W}_{*,n}} \right)
 \end{aligned}$$



$n=0$

# RBM Marginal Distribution

$$\begin{aligned}
 p_n(\mathbf{v}) &\propto e^{-\frac{1}{2}(\mathbf{v}-\mathbf{b})^T(\mathbf{v}-\mathbf{b})} \prod_j \left( 1 + e^{c_j + \mathbf{v}^T \mathbf{W}_{*,j}} \right) \\
 &= p_{n-1}(\mathbf{v}) \left( 1 + e^{c_n + \mathbf{v}^T \mathbf{W}_{*,n}} \right)
 \end{aligned}$$

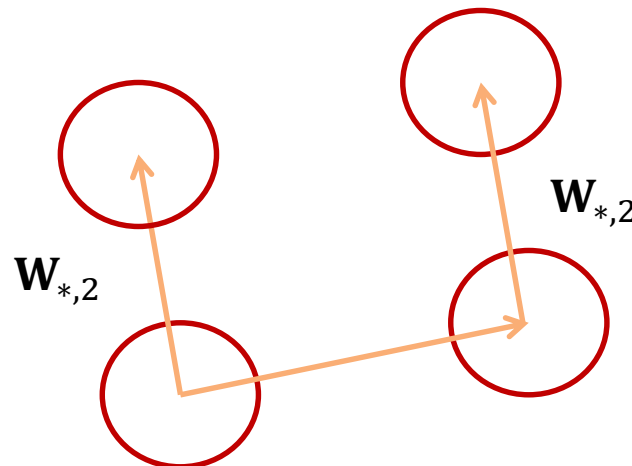


$n=1$

# RBM Marginal Distribution

$$p_n(\mathbf{v}) \propto e^{-\frac{1}{2}(\mathbf{v}-\mathbf{b})^T(\mathbf{v}-\mathbf{b})} \prod_j \left( 1 + e^{c_j + \mathbf{v}^T \mathbf{W}_{*,j}} \right)$$

$$= p_{n-1}(\mathbf{v}) \left( 1 + e^{c_n + \mathbf{v}^T \mathbf{W}_{*,n}} \right)$$

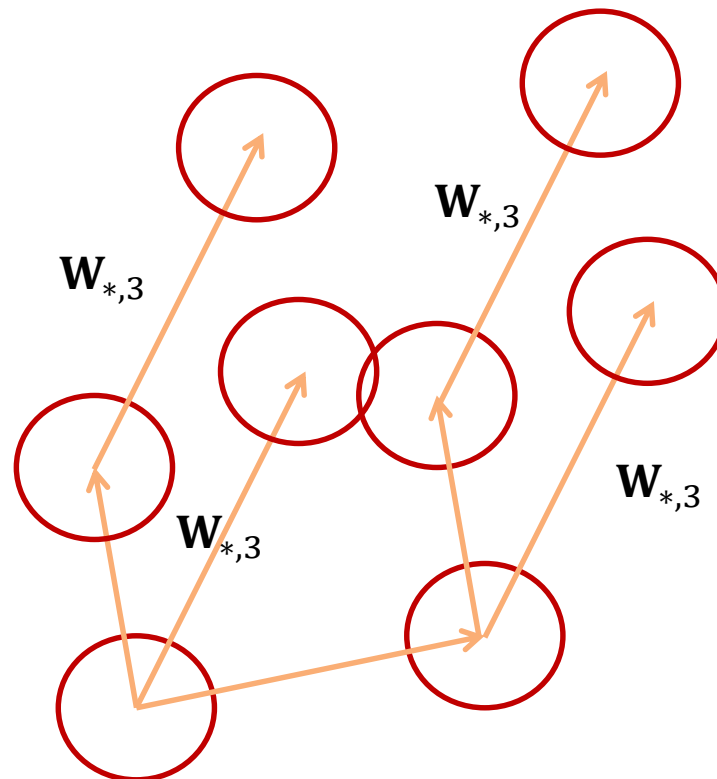


$n=2$

# RBM Marginal Distribution

$$p_n(\mathbf{v}) \propto e^{-\frac{1}{2}(\mathbf{v}-\mathbf{b})^T(\mathbf{v}-\mathbf{b})} \prod_j \left( 1 + e^{c_j + \mathbf{v}^T \mathbf{W}_{*,j}} \right)$$

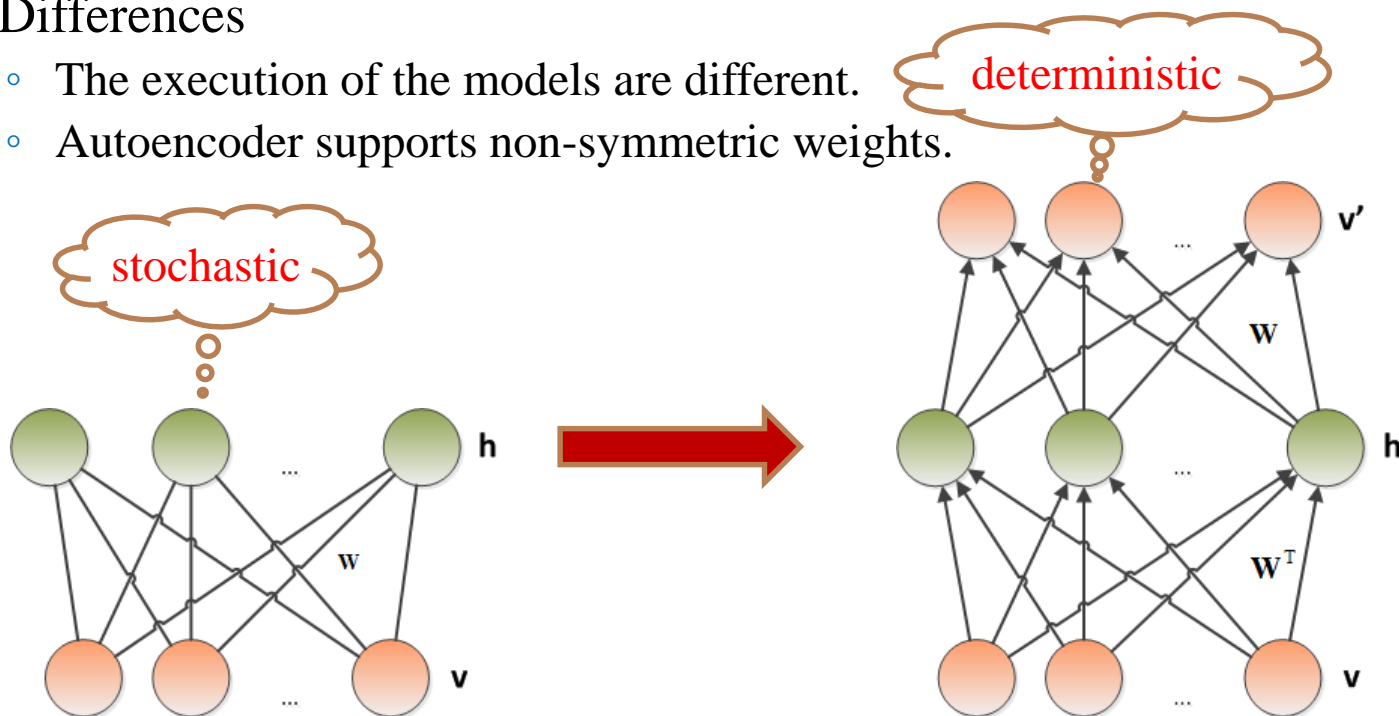
$$= p_{n-1}(\mathbf{v}) \left( 1 + e^{c_n + \mathbf{v}^T \mathbf{W}_{*,n}} \right)$$



$n=3$

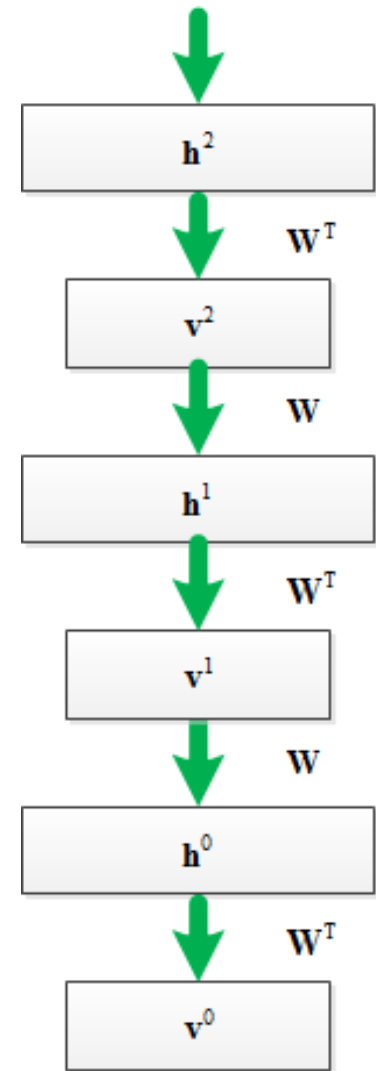
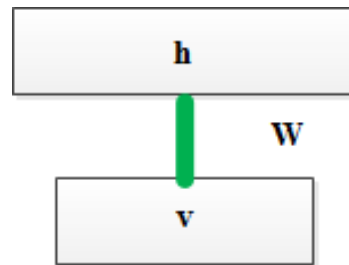
# RBM and Noisy Autoencoder

- Similarities
  - Both can be considered unsupervised feature extractor
  - In both models the hidden layer is the feature
  - RBM can be considered as noisy autoencoder with symmetric weights.
  - The training procedure is similar although they are derived from different angles.
- Differences
  - The execution of the models are different.
  - Autoencoder supports non-symmetric weights.



# RBM and DBN

- An RBM is equivalent to an infinite directed net with replicated weights that define the compatible conditional distributions:  $p(\mathbf{v}|\mathbf{h})$  and  $p(\mathbf{h}|\mathbf{v})$ .
  - A top-down pass of the directed net is exactly equivalent to letting an RBM settle to equilibrium when nothing is clamped.
  - The model above a layer define a complimentary prior for that layer.
  - Inference in the directed net is exactly equivalent to letting an RBM settle to equilibrium starting at the data



# Part 2: Outline

---

- Energy-based model (EBM)
- Restricted Boltzmann machine (RBM)
- **Deep belief network (DBN)**
- Higher-order Boltzmann machines (HOBM)
- Recurrent neural network (RNN)
- Conclusions

# Deep Generative Model

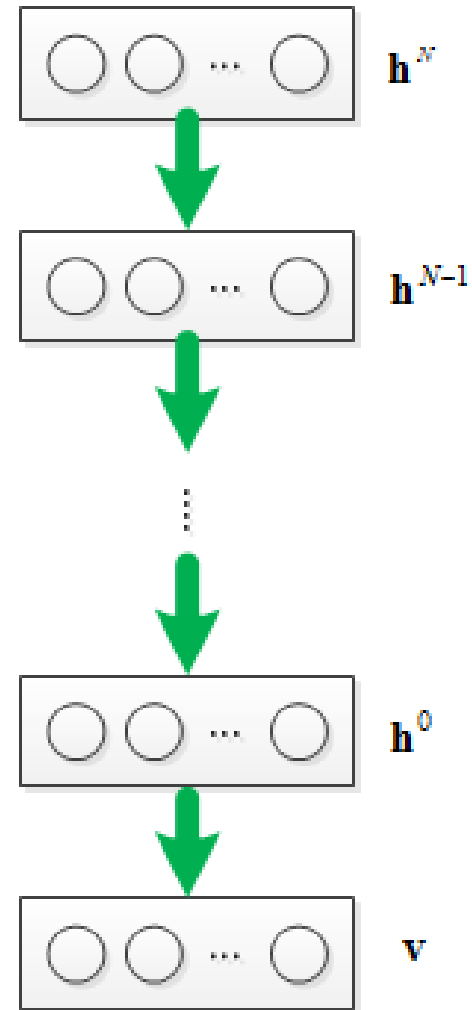
## Intuition

- Deep generative model can model very complicated generation process
- Model learning is very difficult since change in any higher layer will affect all layers below

$$p(\mathbf{v}) = \sum_{\mathbf{h}^0, \mathbf{h}^1, \dots, \mathbf{h}^N} p(\mathbf{h}^N) \left( \prod_{n=0}^{N-1} p(\mathbf{h}^n | \mathbf{h}^{n+1}) \right) p(\mathbf{v} | \mathbf{h}^0)$$

Hidden Representation  
Can be universal

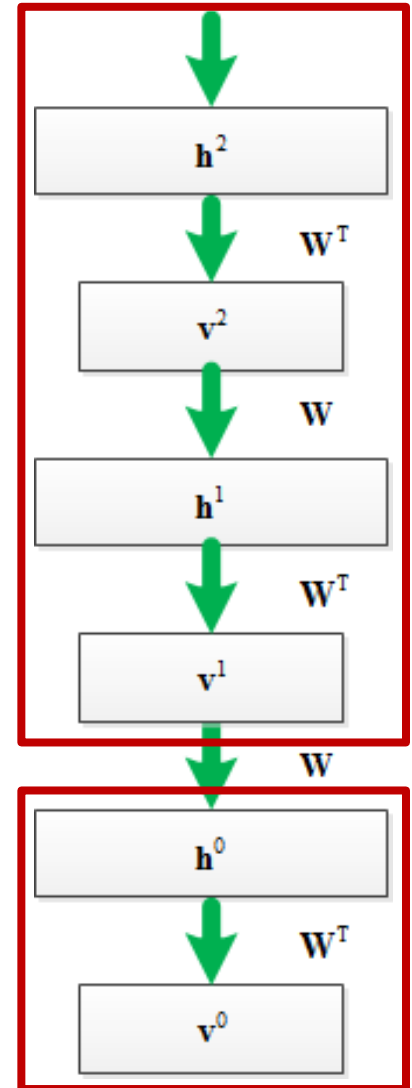
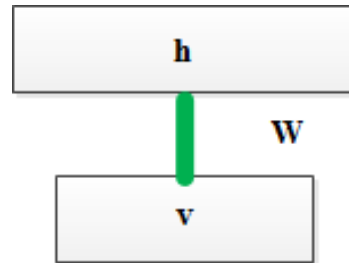
Visible Observation  
Application Dependent



# Learning a DBN: One Way

- First learn with all the weights tied
  - equivalent to learning an RBM
- Then freeze the first layer of weights and learn the remaining weights (still tied together).
  - equivalent to learning another RBM, using the aggregated conditional probability on  $\mathbf{h}_0$  as the data
  - Continue the process to train the next layer
- Intuitively  $\log p(\mathbf{v})$  improves as new layer is added and trained.

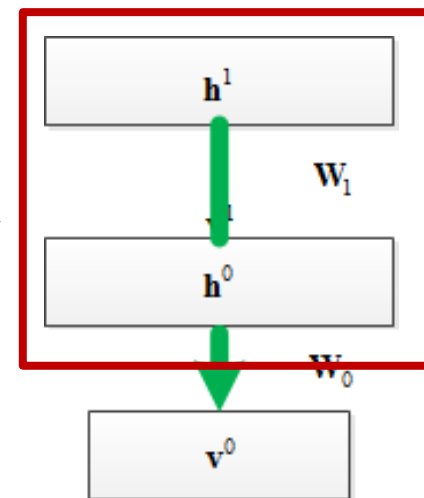
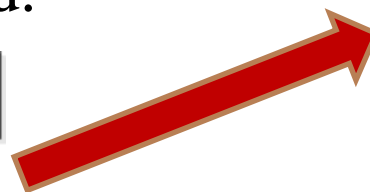
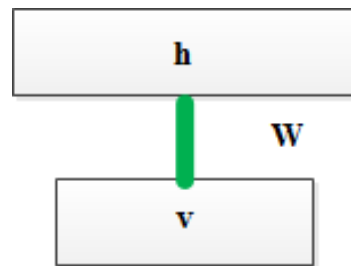
It's Sub optimal!



# Learning a DBN: One Way

- First learn with all the weights tied
  - equivalent to learning an RBM
- Then freeze the first layer of weights and learn the remaining weights (still tied together).
  - equivalent to learning another RBM, using the aggregated conditional probability on  $\mathbf{h}_0$  as the data
  - Continue the process to train the next layer
- Intuitively  $\log p(\mathbf{v})$  improves as new layer is added and trained.

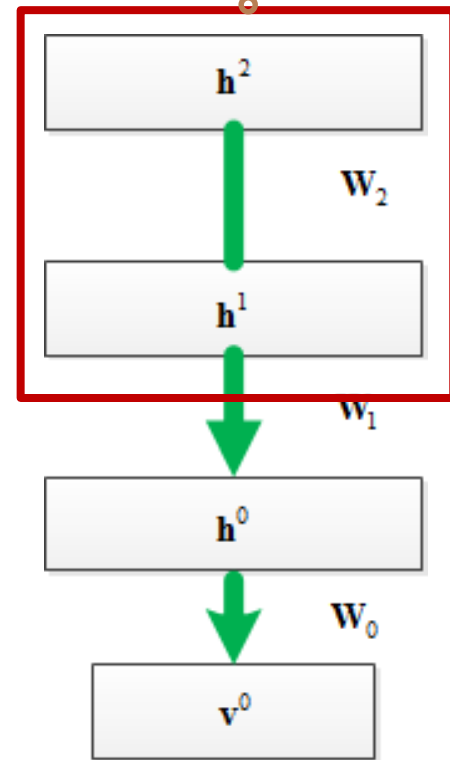
It's Sub  
optimal!



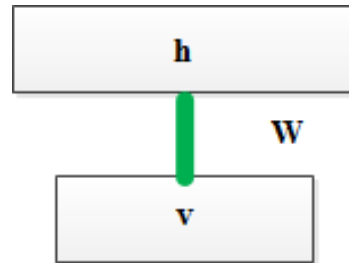
# Learning a DBN: One Way

- First learn with all the weights tied
  - equivalent to learning an RBM
- Then freeze the first layer of weights and learn the remaining weights (still tied together).
  - equivalent to learning another RBM, using the aggregated conditional probability on  $\mathbf{h}_0$  as the data
  - Continue the process to train the next layer
- Intuitively  $\log p(\mathbf{v})$  improves as new layer is added and trained.

If perfectly trained:  
 $h^3 = h^1 = v$

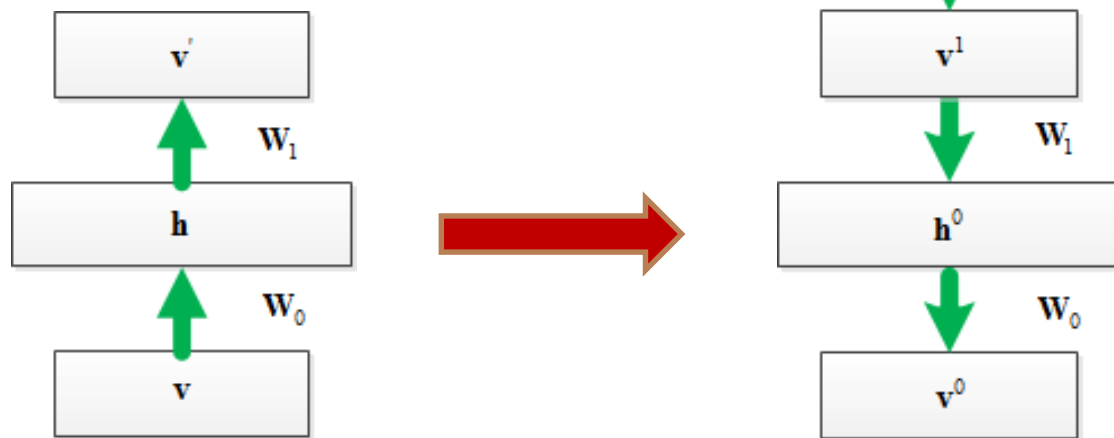


It's Sub  
optimal!



# Learning a DBN: Alternative Way

- Use the same greedy layer by layer learning algorithm
- But train each layer with noisy autoencoder
- More flexible than RBM since weights no need to be tied
- Performs similar to RBM based approach



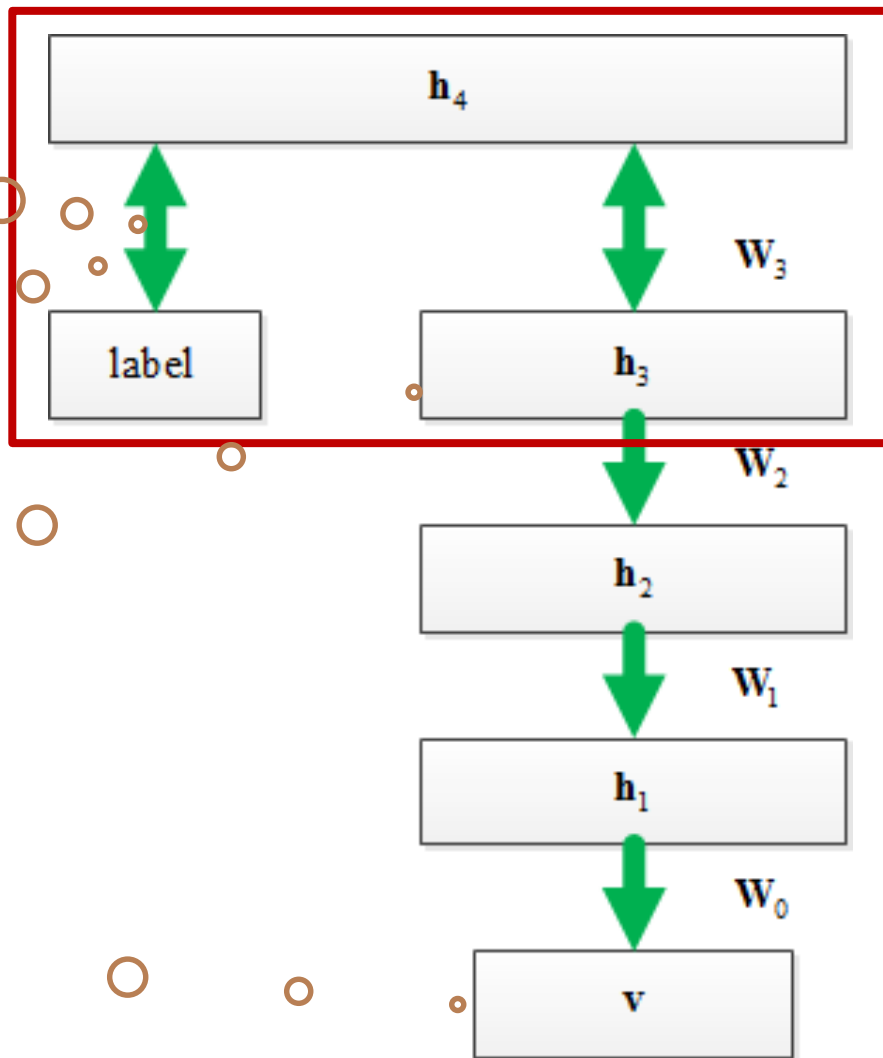
# How to Use DBN: Generation

learns to generate combinations of labels and features

1. Run the top layer to thermal equilibrium with or without label clamped

2A. Sample from the distribution and then top-down till end

2B. Calculate  $p(\mathbf{v})$  and sample from it.



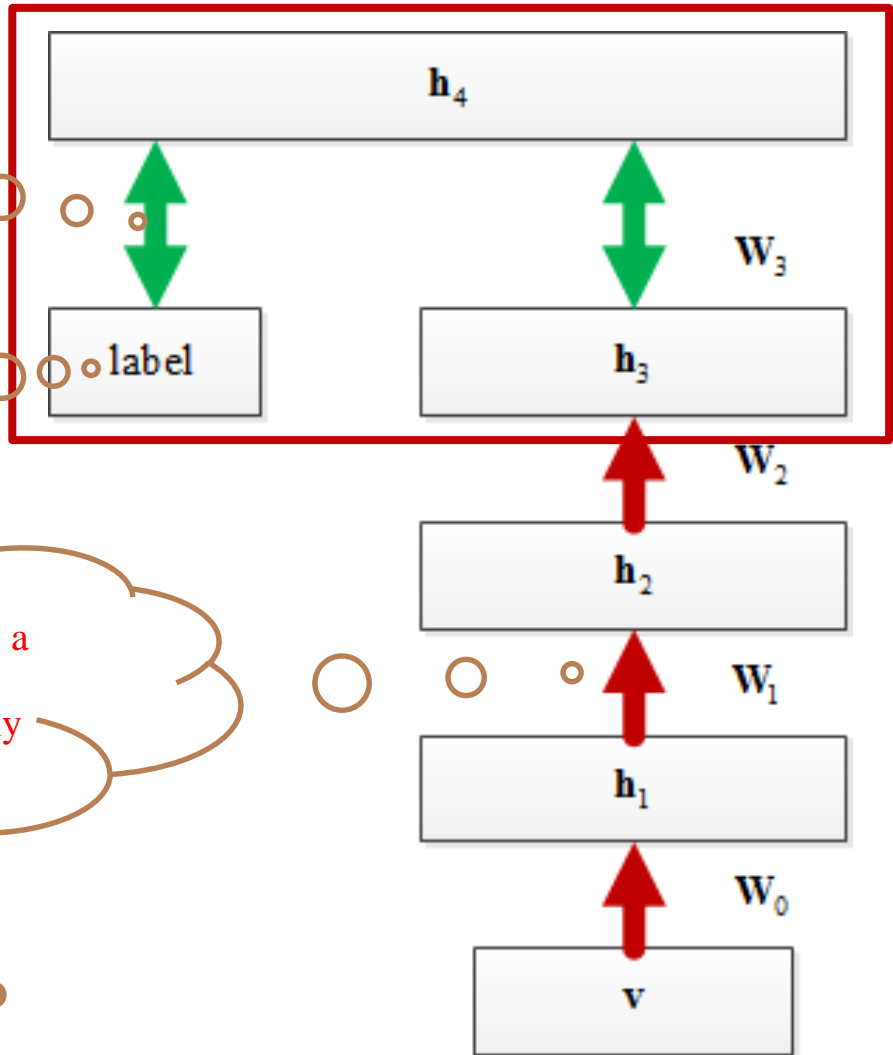
# How to Use DBN: Recognition

learns to generate combinations of labels and features

1. Set label to neutral state (needed if we want to speed up)

2. Run up-forward and either follow a few iterations of the top RBM or calculate the label probability directly as in the discriminative RBM

3. Pick the label with highest probability



# Fine Tuning: Generative

- Greedy layer by layer learning is sub-optimal. The performance can be improved with fine-tuning
- “wake-sleep” algorithm (**very slow**)
  1. Do a stochastic bottom-up pass
    - Adjust the top-down weights to be good at reconstructing the feature activities in the layer below.
  2. Do a few iterations of sampling in the top level RBM
    - -- Adjust the weights in the top-level RBM.
  3. Do a stochastic top-down pass
    - Adjust the bottom-up weights to be good at reconstructing the feature activities in the layer above.
- After fine-tuning, weights are no longer tied

# Part 2: Outline

---

- Energy-based model (EBM)
- Restricted Boltzmann machine (RBM)
- Deep belief network (DBN)
- **Higher-order Boltzmann machines (HOBM)**
- Recurrent neural network (RNN)

# Three-Way Boltzmann Machine

- The normal RBM only contains pairwise interactions

$$E(\mathbf{v}, \mathbf{h}; \theta) = - \sum_{i,j} w_{ij} v_i h_j - \text{bias\_term}$$

- But higher-order interactions might be useful to describe the generation process

$$E(\mathbf{v}, \mathbf{h}; \theta) = - \sum_{i,j,k} w_{ijk} v_i v_j h_k$$

- Too many parameters !!!



# Factored Three-Way RBM

- Reduce three-way weights to two-way weights by introducing factors

$$w_{ijk} = \sum_f B_{if} C_{jf} P_{kf}$$

- Further ties  $B$  and  $C$  and get:  $w_{ijk} = \sum_f C_{if} C_{jf} P_{kf}$

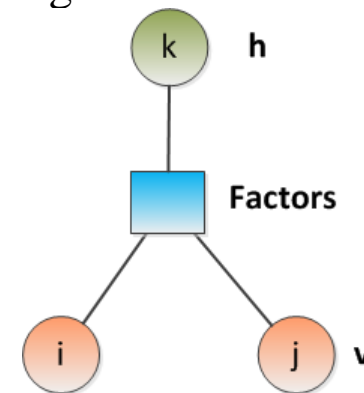
- Energy:  $E(\mathbf{v}, \mathbf{h}; \theta) = -\sum_f \left( \sum_i v_i C_{if} \right)^2 \left( \sum_k h_k P_{kf} \right)$

- Posterior:  $p(h_k = 1 | \mathbf{v}) = \sigma \left( \sum_f P_{kf} \left( \sum_i v_i C_{if} \right)^2 + b_k \right)$

- Free energy:  $F(v) = -\sum_k \log \left( 1 + \exp \left( \sum_f P_{kf} \left( \sum_i v_i C_{if} \right)^2 + b_k \right) \right)$

- Hybrid Monte Carlo algorithm (HMC)**

draws a sample by simulating a particle moving on the free energy surface. The particle starts at the data-point and is given an initial random momentum sampled from a spherical Gaussian with unit variance. It then moves over the surface using the gradient of the free energy to determine its acceleration. If the sum of the potential do not change, accept the results. If the total energy rises during the simulation, the result is accepted with a probability equal to the negative exponential of the total energy increase.



# RBM Conditioned on Visible

- A generative model for time series data
- Defines a joint probability distribution over  $\mathbf{v}_t$  and  $\mathbf{h}_t$ , conditional on  $\mathbf{v}_{<t}$ .

- $$p(\mathbf{v}_t, \mathbf{h}_t | \mathbf{v}_{<t}, \theta) = \frac{\exp(-E(\mathbf{v}_t, \mathbf{h}_t | \mathbf{v}_{<t}, \theta))}{Z}$$

- $$E = \sum_i \frac{(\hat{a}_{i,t} - v_{i,t})^2}{2\sigma_i^2} - \sum_j \hat{b}_{j,t} h_{j,t} - \sum_{i,j} W_{ij} \frac{v_{i,t}}{\sigma_i} h_{j,t}$$

net input from the past predicts the visible

- $$\hat{a}_{i,t} = a_i + \sum_k A_{ki} v_{k,<t}$$

- $$\hat{b}_{j,t} = b_j + \sum_k B_{kj} v_{k,<t}$$

net input from the past defines the hidden

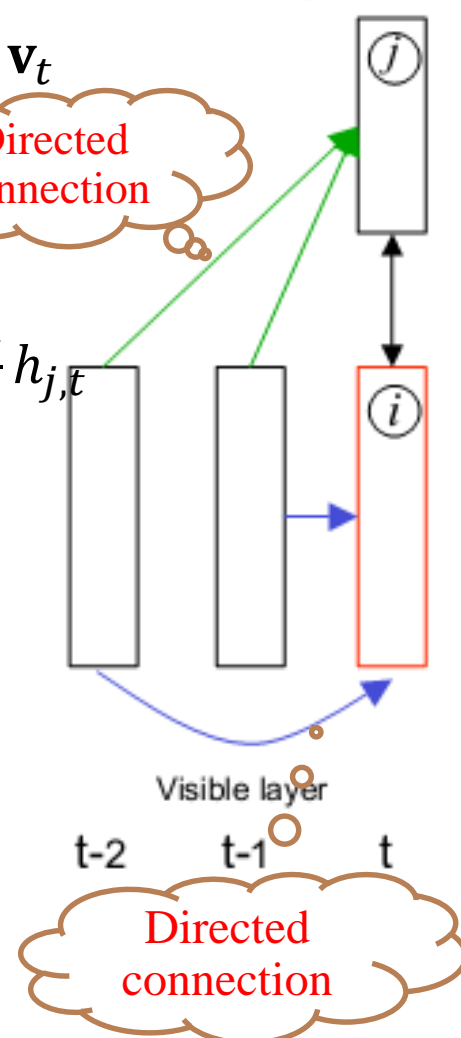
Hidden layer

Directed connection

Visible layer

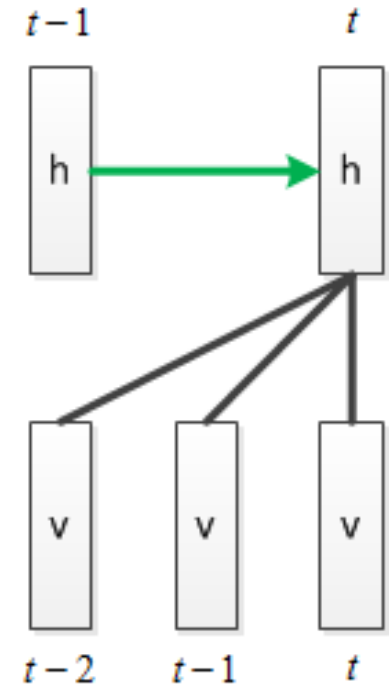
t-2 t-1 t

Directed connection



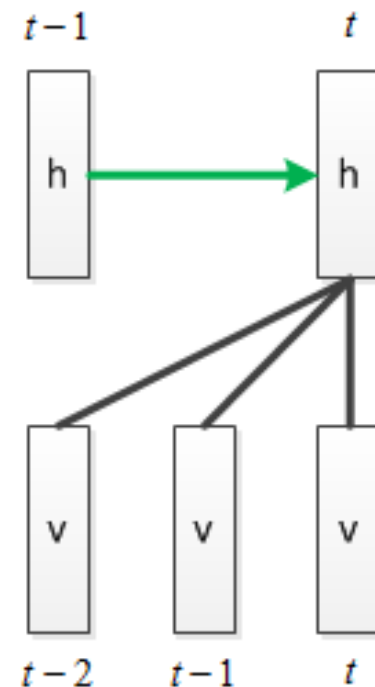
# RBM Conditioned on Hidden

- Given the data and the previous hidden state, the hidden units at time  $t$  are conditionally independent.
- The temporal connections between hidden units can be treated as additional biases
- Learning can be done by using contrastive divergence.
  - First learn a static model ignoring the directed temporal connections between hidden units.
  - Then use the inferred hidden states to train a “fully observed” sigmoid belief net that captures the temporal structure of the hidden states.
  - Finally, fine tune all of the weights.



# Generating from CRBM

- Keep the previous hidden and visible states fixed
  - They provide a time-dependent bias for the hidden units.
- Perform alternating Gibbs sampling for a few iterations between the hidden units and the current visible units.
  - This picks new hidden and visible states that are compatible with each other and with the recent history.
- Only approximate because it ignores the future.
- The model is exponentially more powerful than an HMM because it uses distributed representations.
  - Given  $N$  hidden units, it can use  $N$  bits of information to constrain the future.



# Part 2: Outline

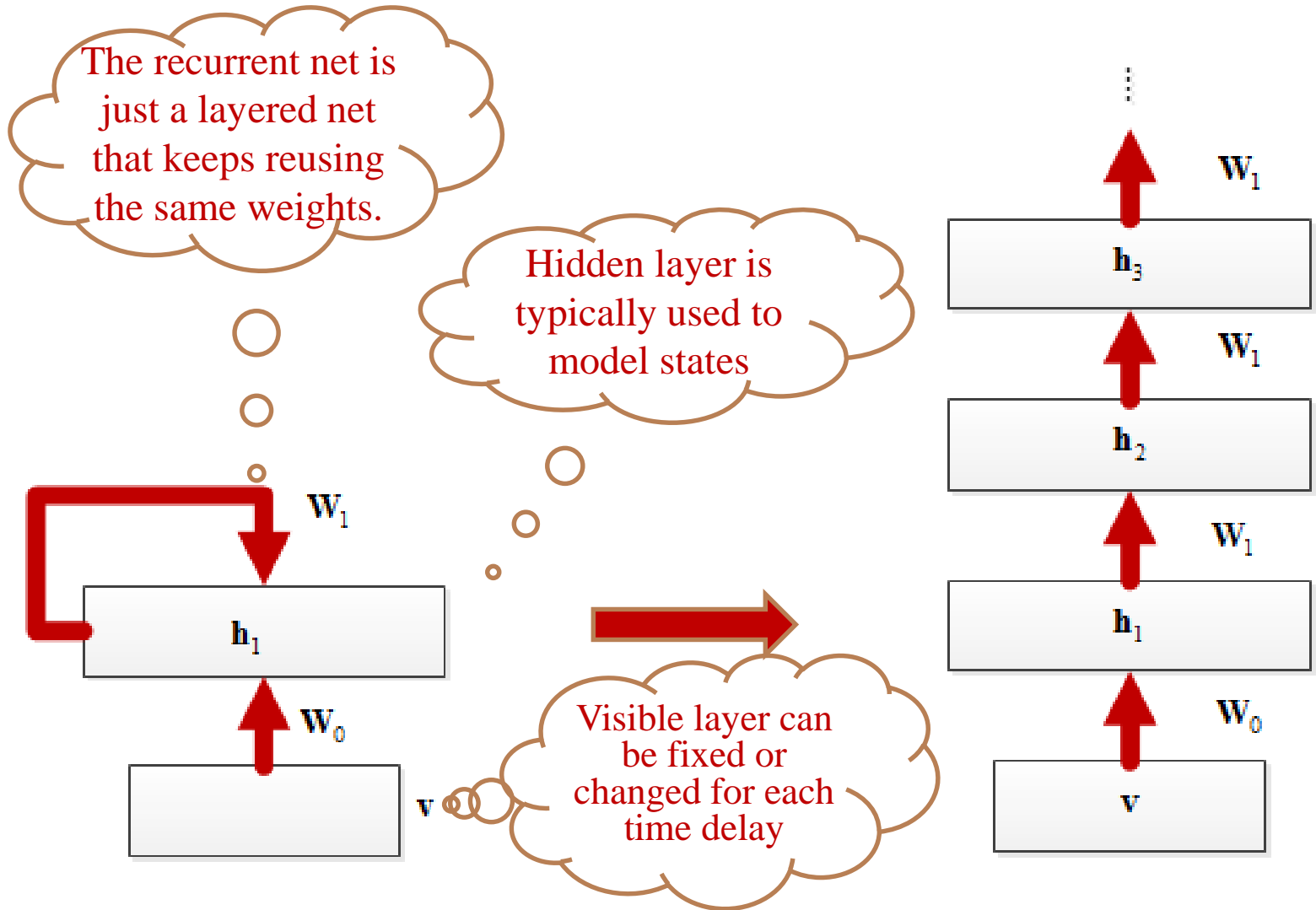
---

- Energy-based model (EBM)
- Restricted Boltzmann machine (RBM)
- Deep belief network (DBN)
- Higher-order Boltzmann machines (HOBM)
- **Recurrent neural network (RNN)**

# Recurrent Neural Network

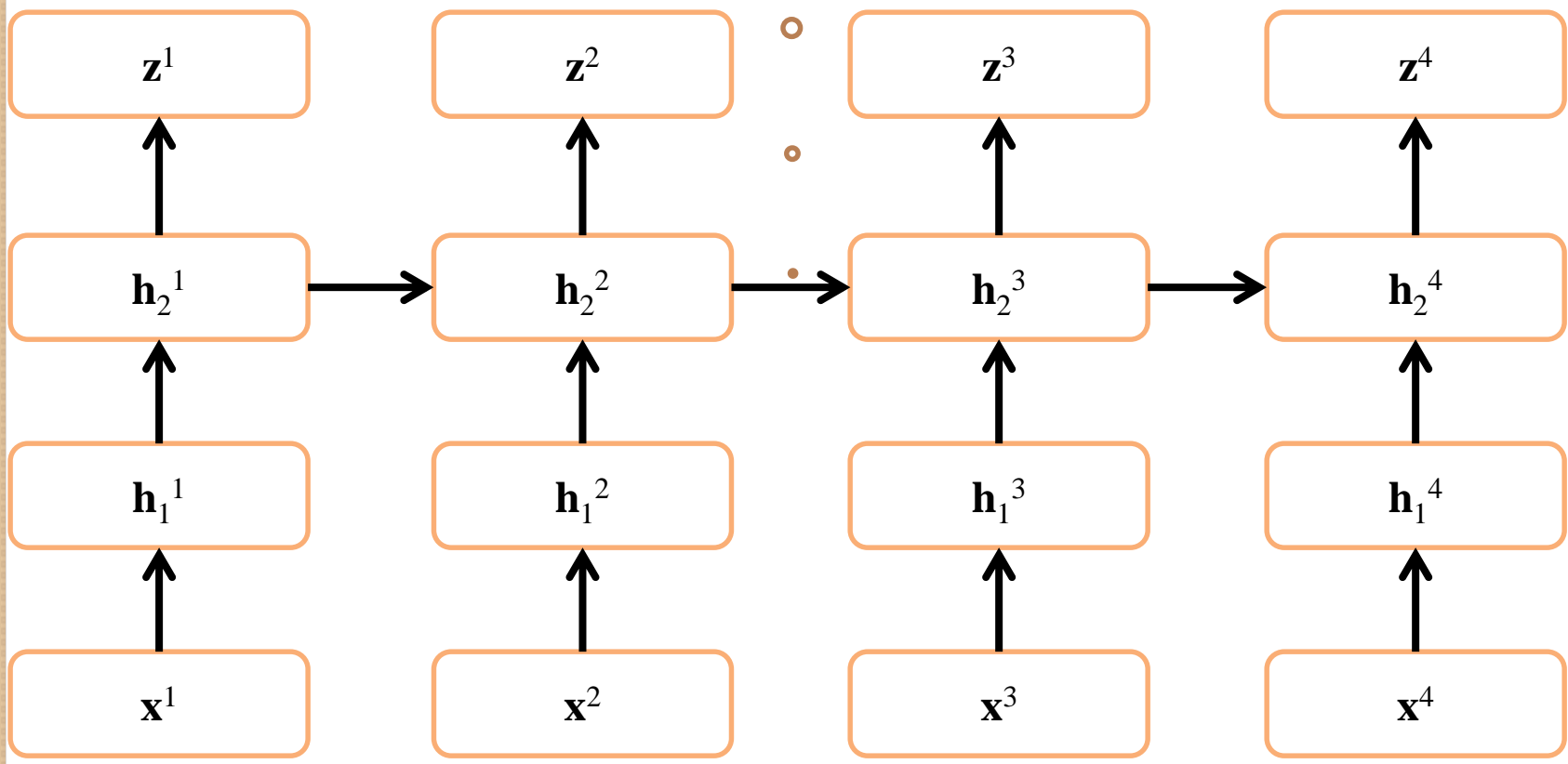
- If the connectivity has directed cycles, the network can do much more than just computing a fixed sequence of non-linear transforms:
  - It can oscillate.
  - It can settle to point attractors.
    - Good for classification
  - It can behave chaotically
    - usually a bad idea for information processing.
  - It can remember things for a long time.
    - The network has internal state. It can decide to ignore the input for a while if it wants to.
  - It can model sequential data in a natural way.
    - No need to use delay taps to spatialize time.

# Recurrent Neural Network



# Recurrent Neural Network

Link over time.

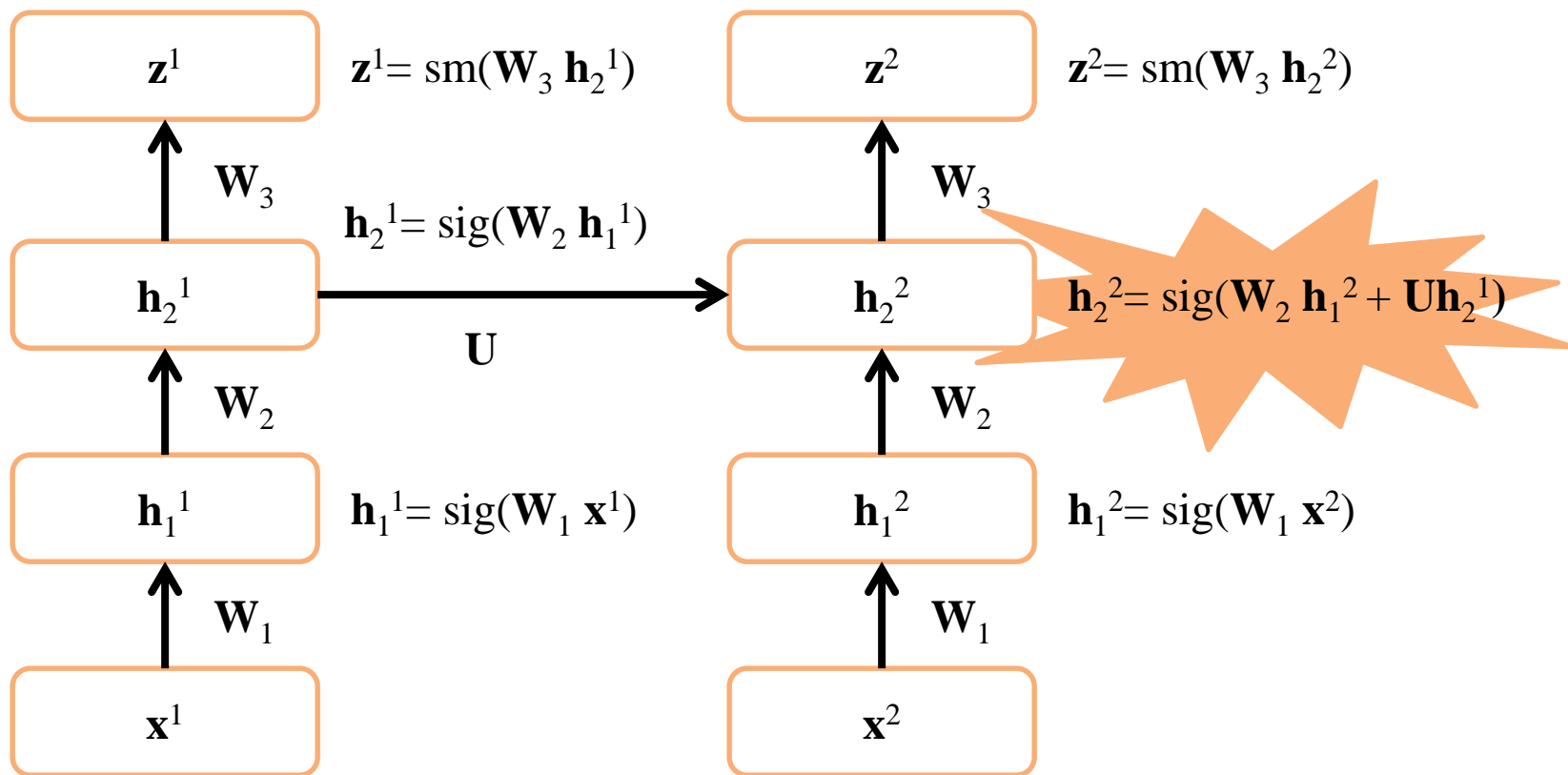


EBM | RBM | DBN | HOBM | **RNN**

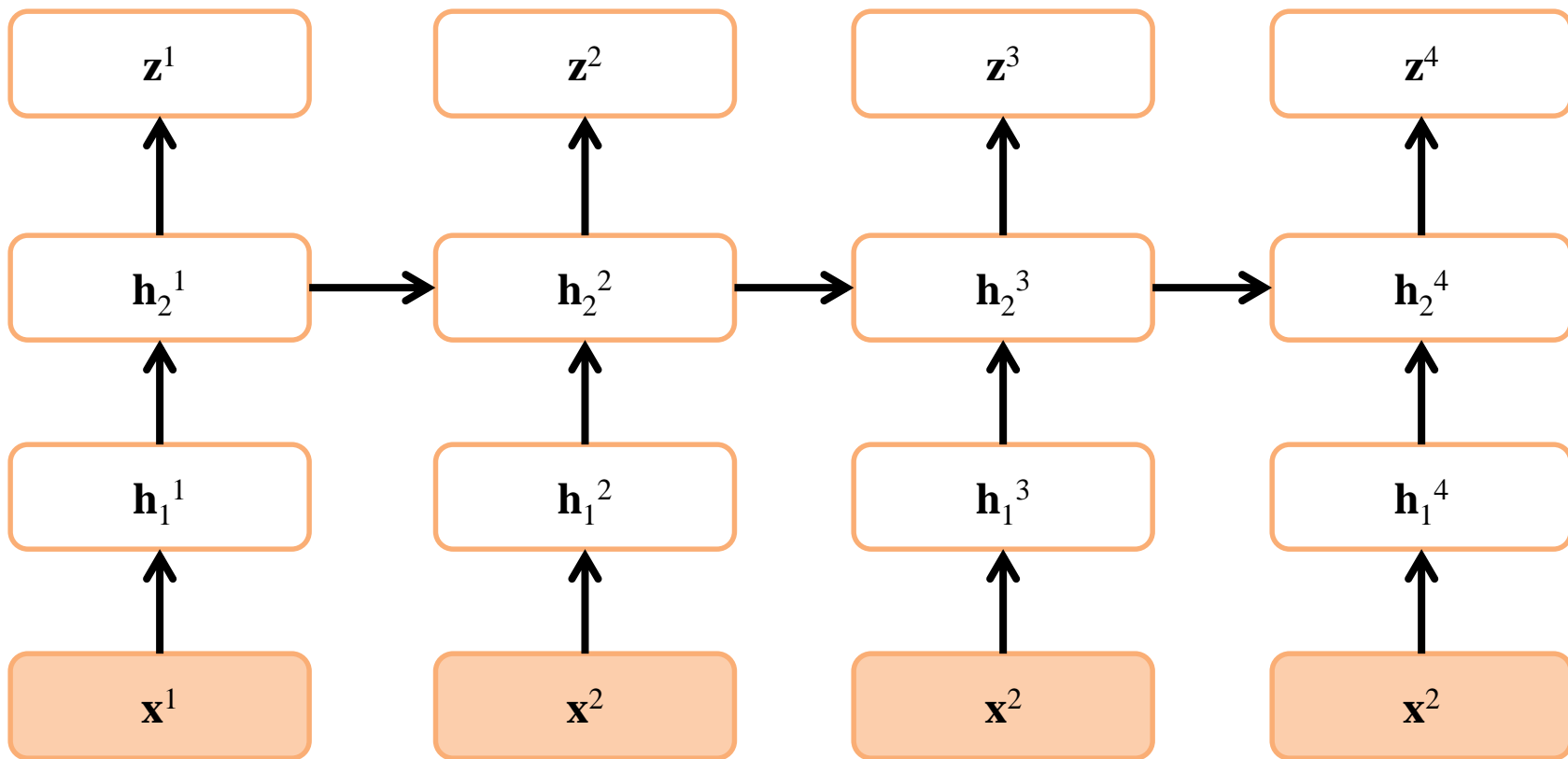
# Model Learning

- Possible targets in RNN
  - Desired final activities of all the units
  - Desired activities of all units for the last few steps
  - Desired activity of a subset of the units.
    - The other units are “hidden”
- Backpropagation through time
  - forward pass: builds up a stack of the activities of all the units at each time step.
  - backward pass: peels activities off the stack to compute the error derivatives at each time step.
  - Summation pass: add together the derivatives at all the different times for each weight.
- Optimization algorithms
  - Quasi-Newton: Exact minimization on a very crude quadratic approximation to the curvature.
  - Hessian-Free: partial minimization on a much better quadratic approximation to the curvature

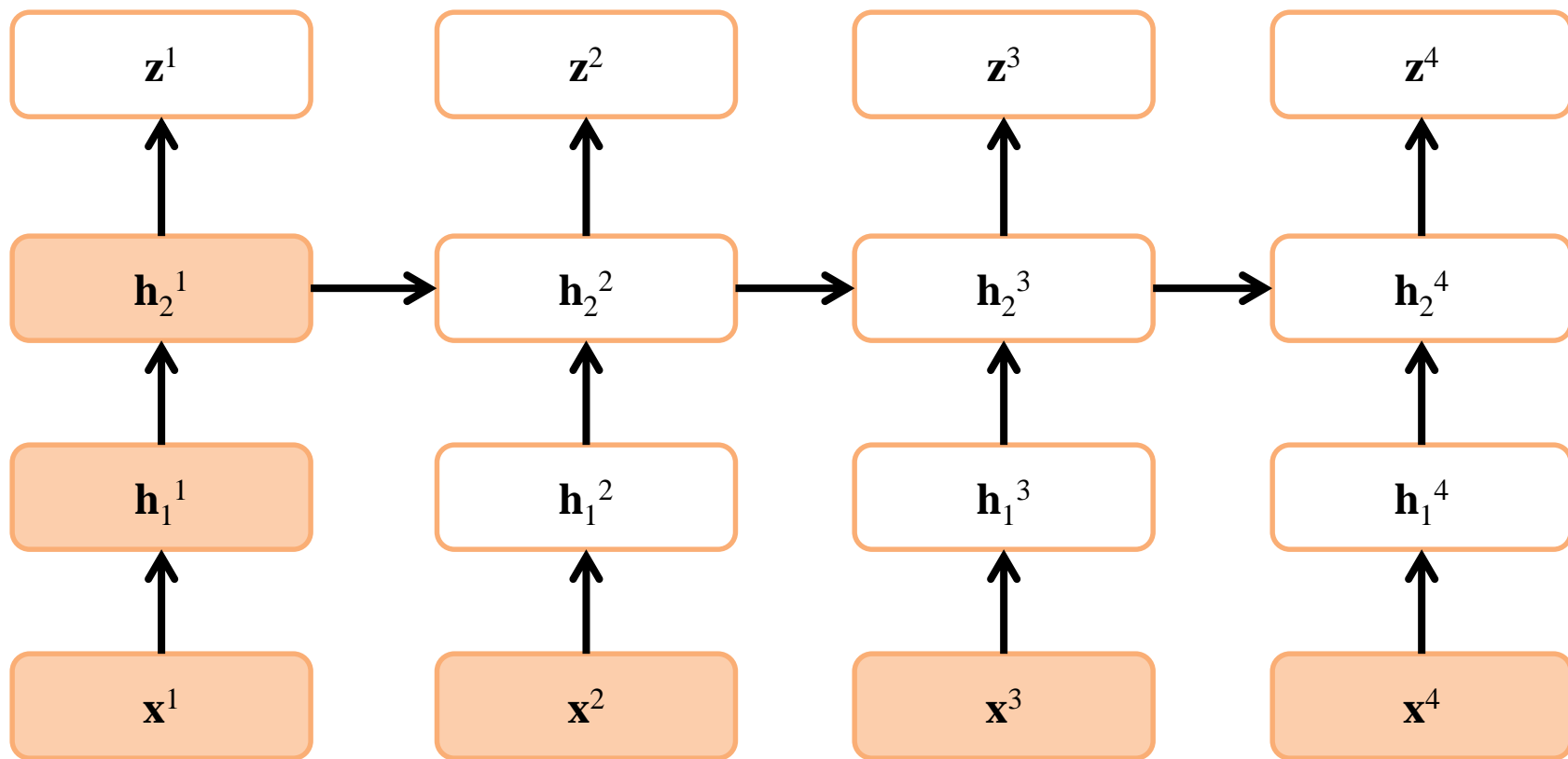
# Sequence Propagation



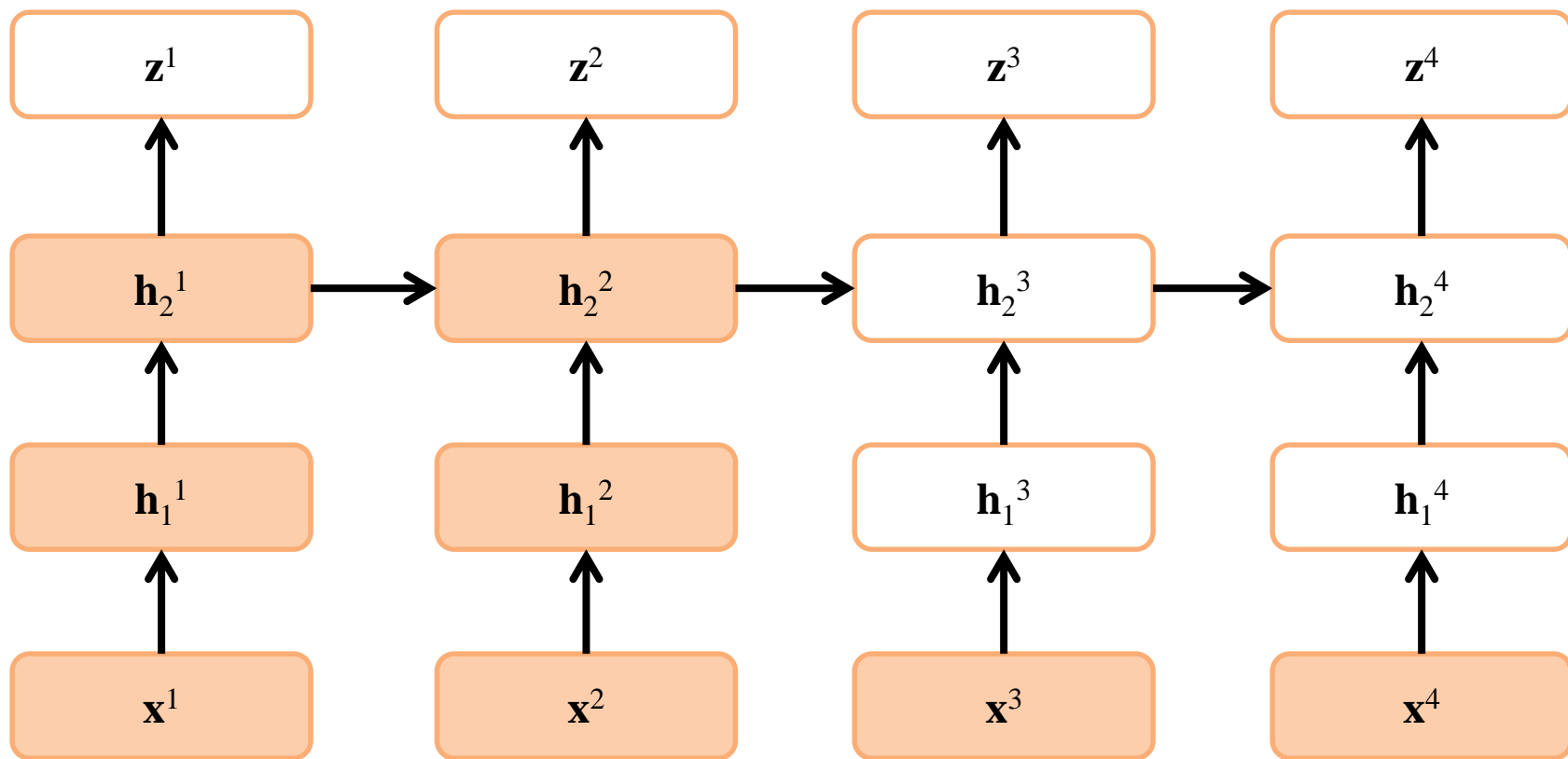
# Sequence Propagation



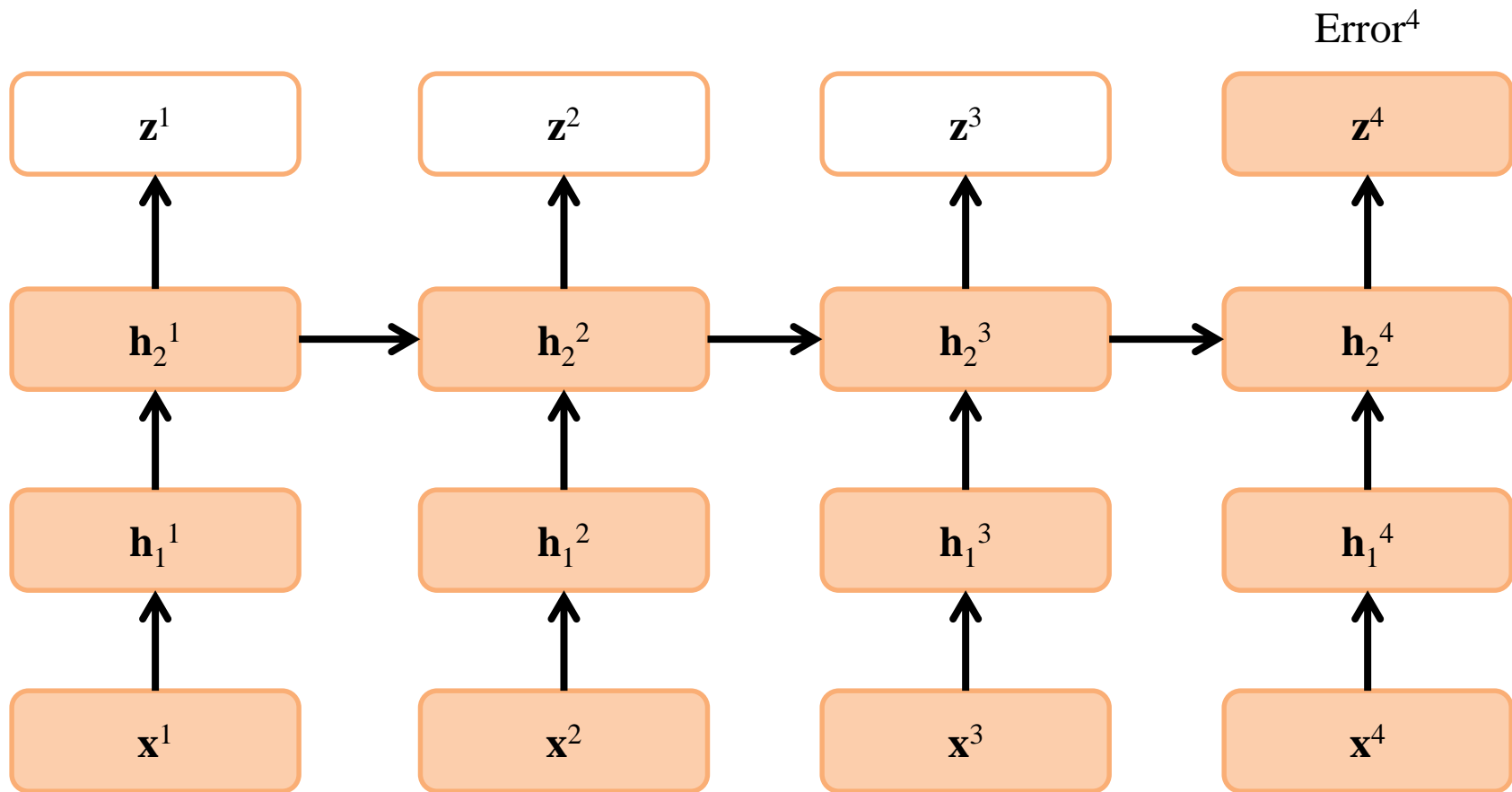
# Sequence Propagation



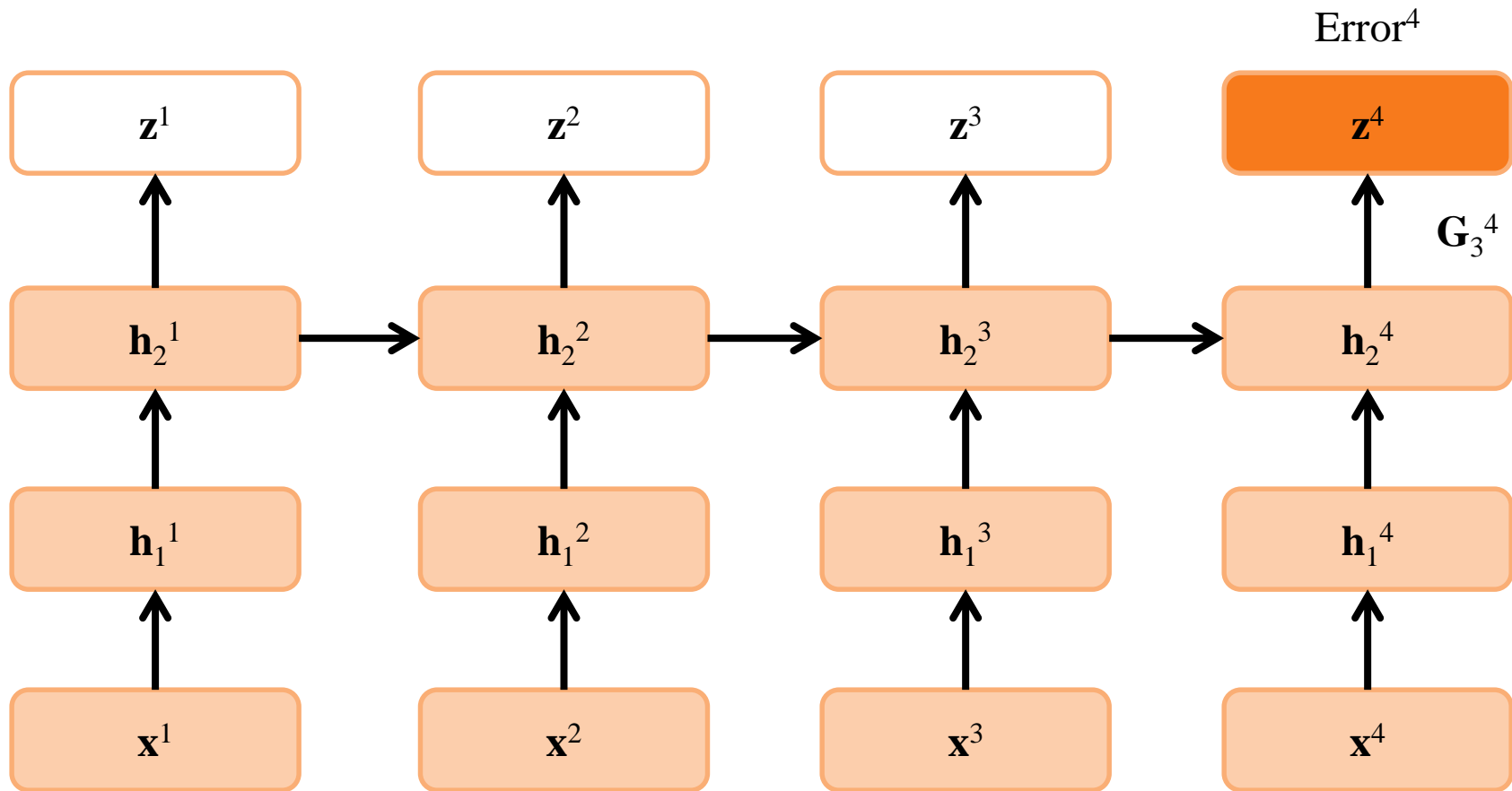
# Sequence Propagation



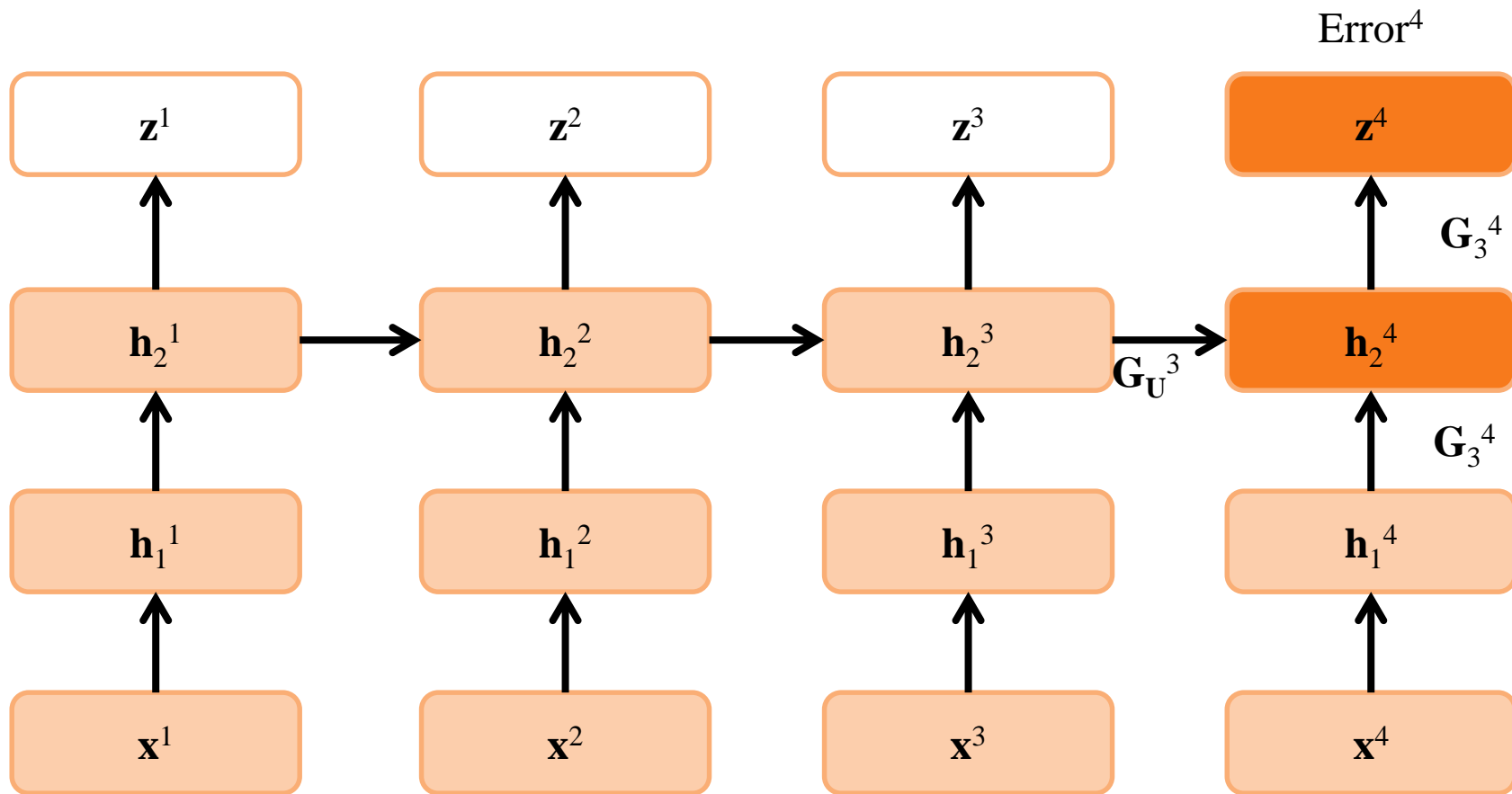
# Sequence Propagation



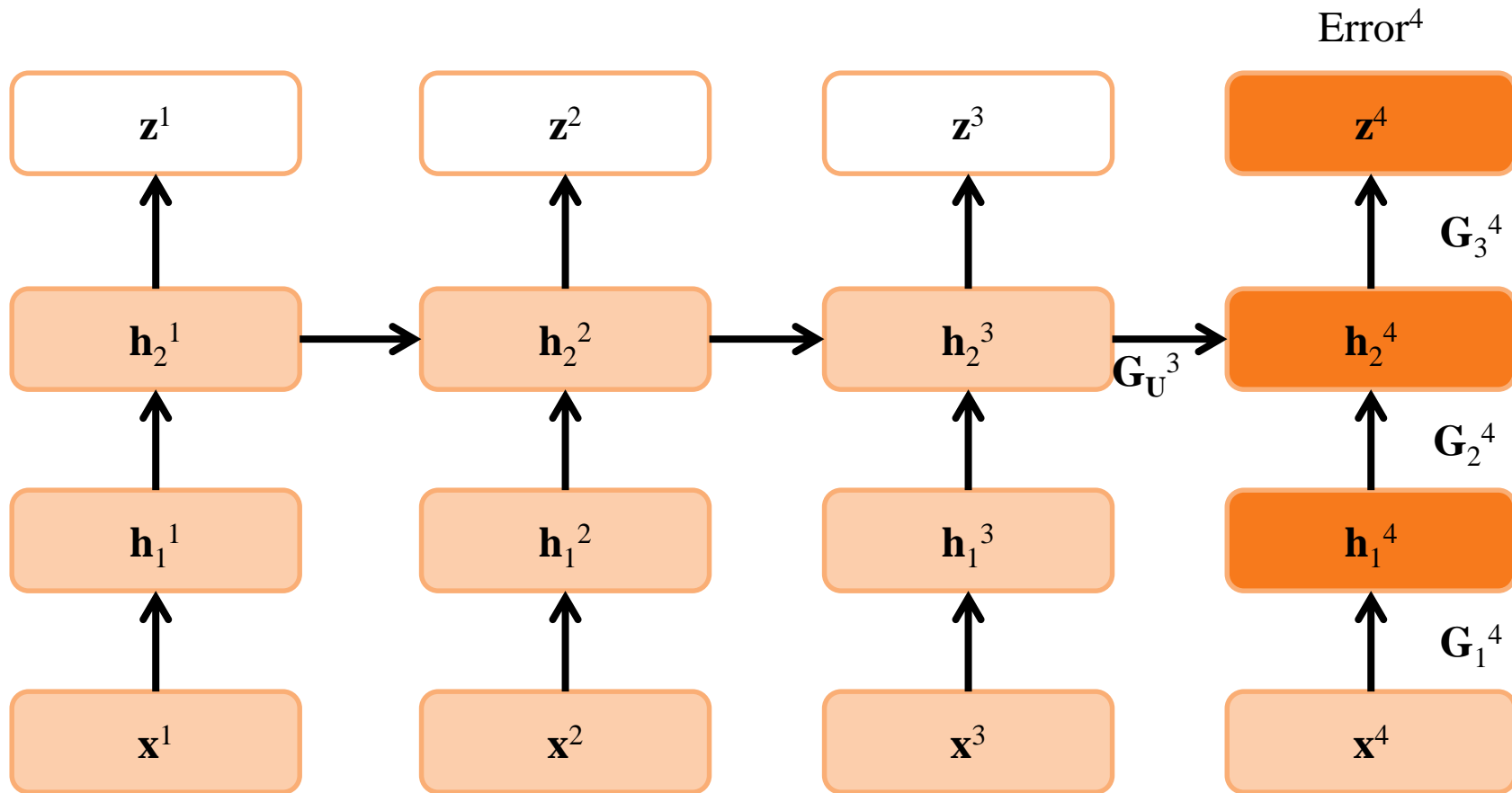
# Gradient Computation



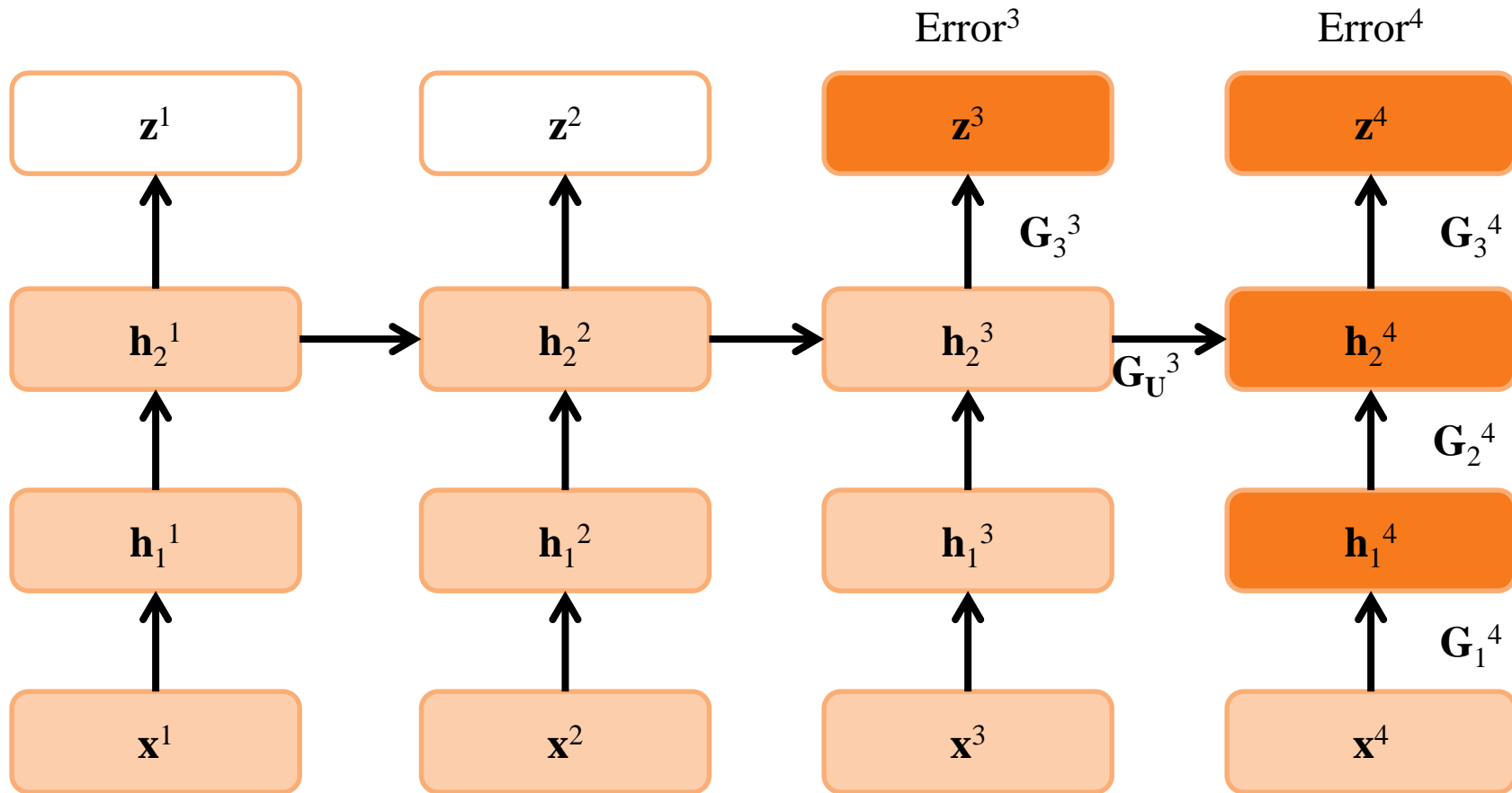
# Gradient Computation



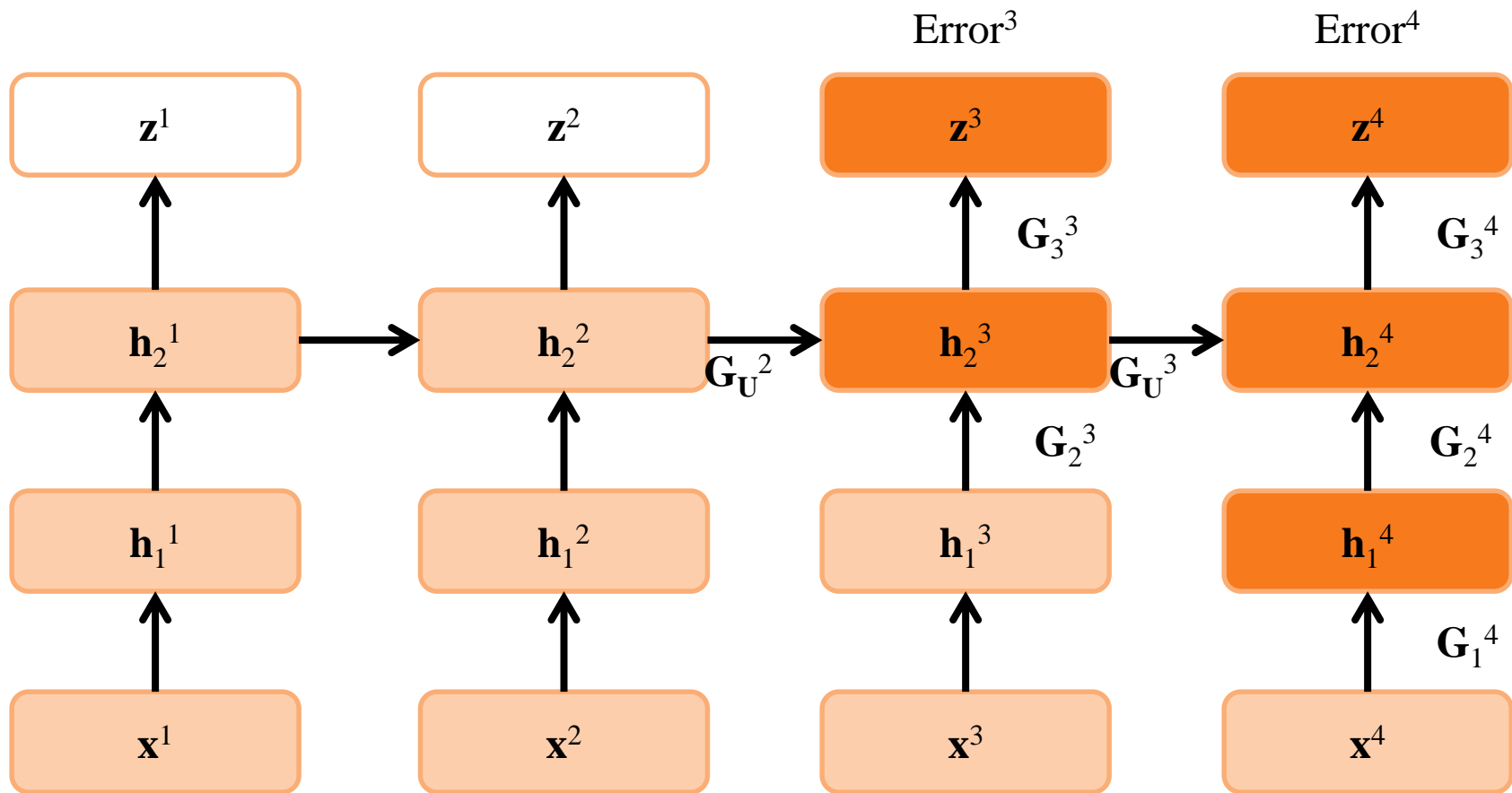
# Gradient Computation



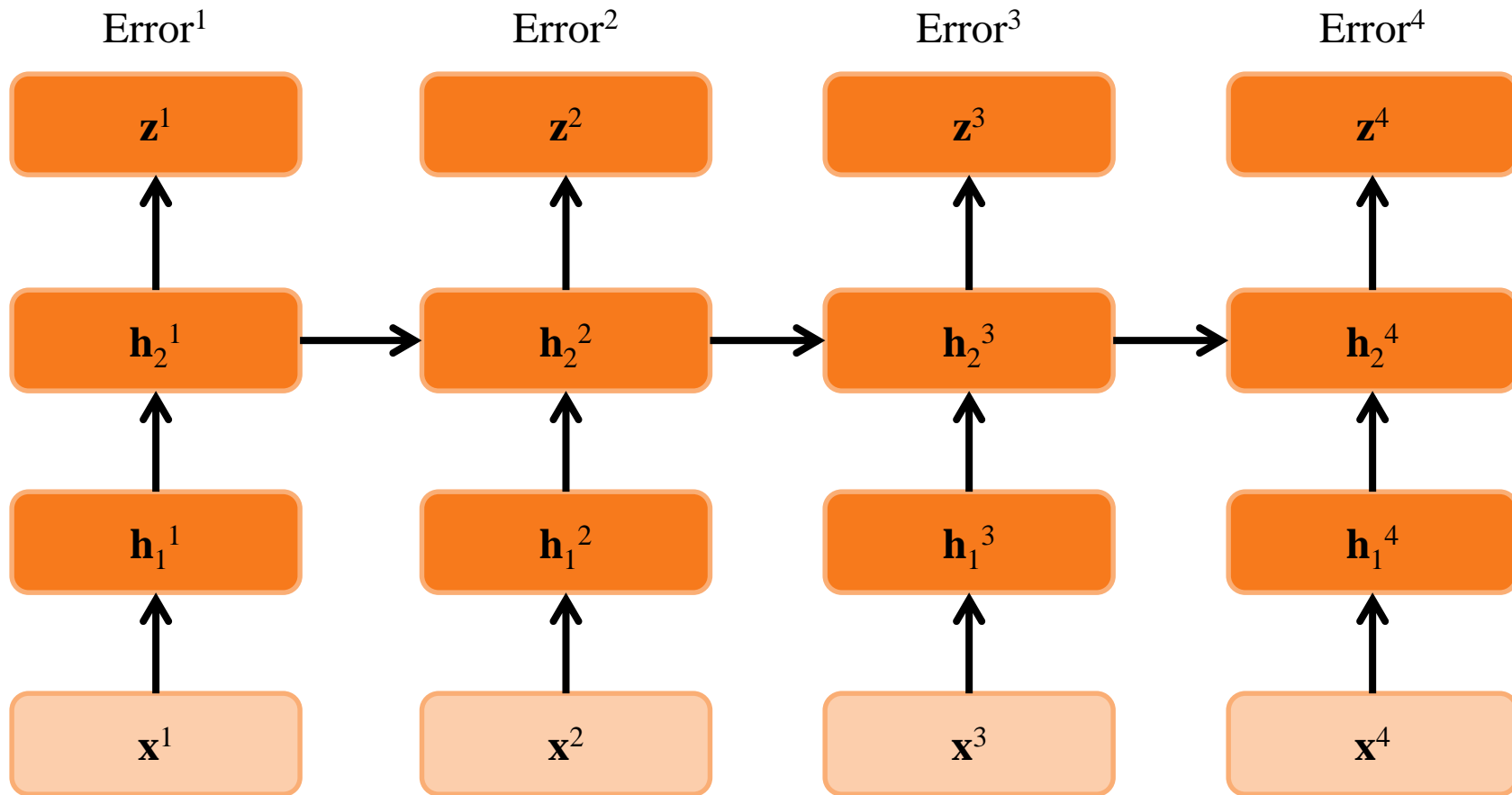
# Gradient Computation



# Gradient Computation



# Gradient Computation



# Other Interesting Models

---

- Not covered but interesting models
  - Deep Boltzmann Machine
  - Hierarchical Temporal Memory

# Questions?

