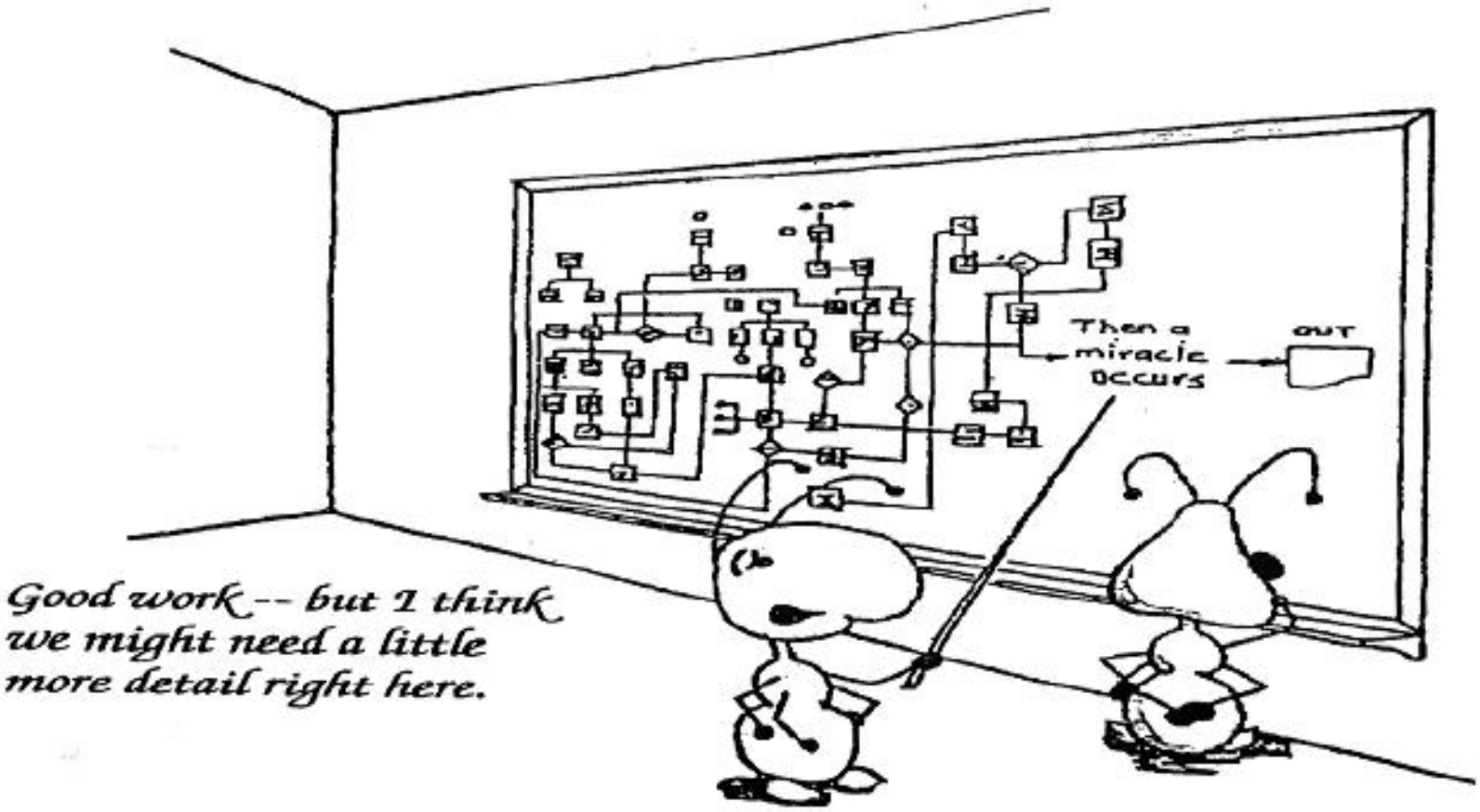
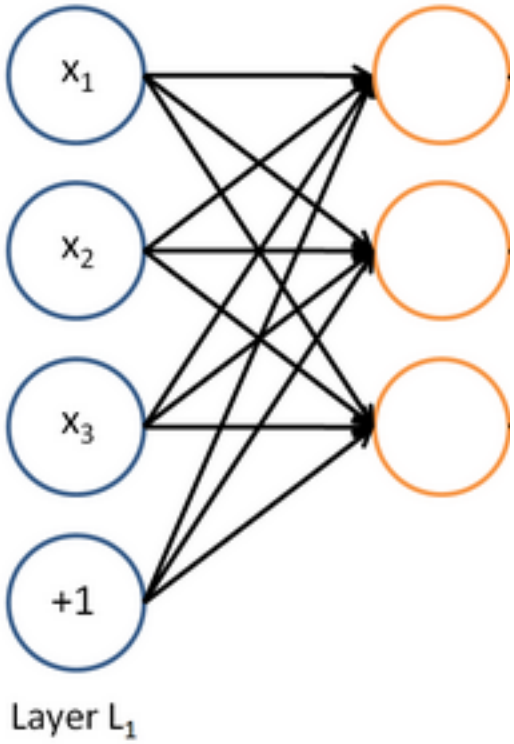

Then a miracle occurs
OUT
Good work -- but I think we might need a little more detail right here.

Part 2

Representation Learning Algorithms

A neural network = running several logistic regressions at the same time

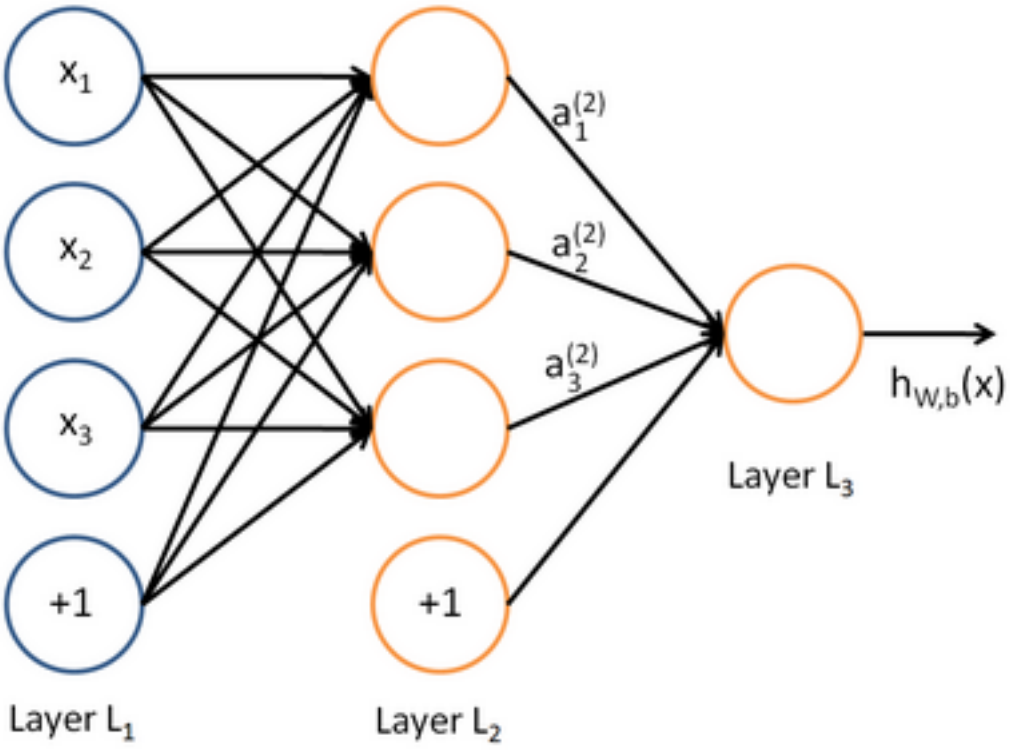
If we feed a vector of inputs through a bunch of logistic regression functions, then we get a vector of outputs



But we don't have to decide ahead of time what variables these logistic regressions are trying to predict!

A neural network = running several logistic regressions at the same time

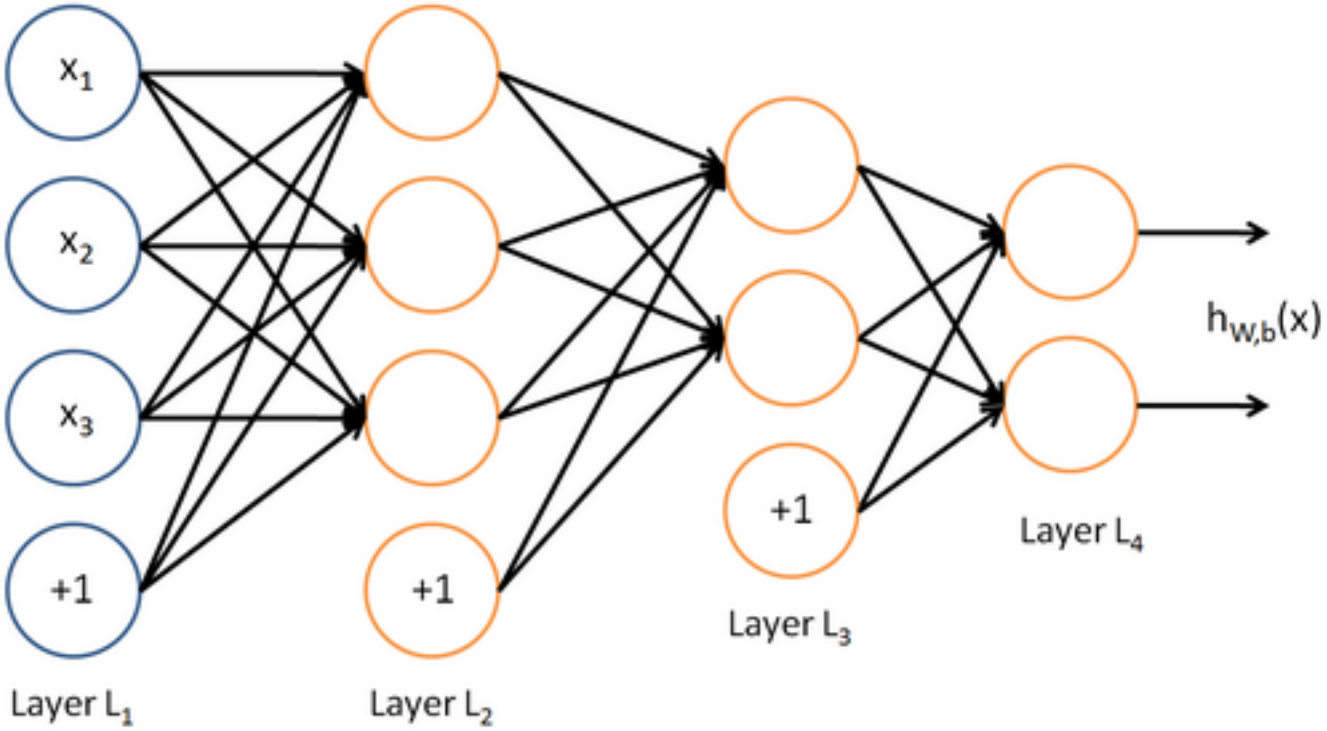
… which we can feed into another logistic regression function



and it is the training criterion that will decide what those intermediate binary target variables should be, so as to make a good job of predicting the targets for the next layer, etc.

A neural network = running several logistic regressions at the same time

- Before we know it, we have a multilayer neural network….



Back-Prop

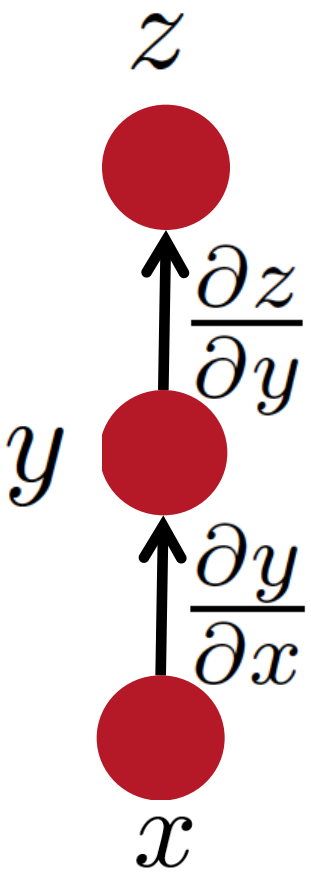
- Compute gradient of example-wise loss wrt parameters

- Simply applying the derivative chain rule wisely

$$z = f(y) \quad y = g(x) \quad \frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}$$

- If computing the loss(example, parameters) is $O(n)$ computation, then so is computing the gradient

Simple Chain Rule



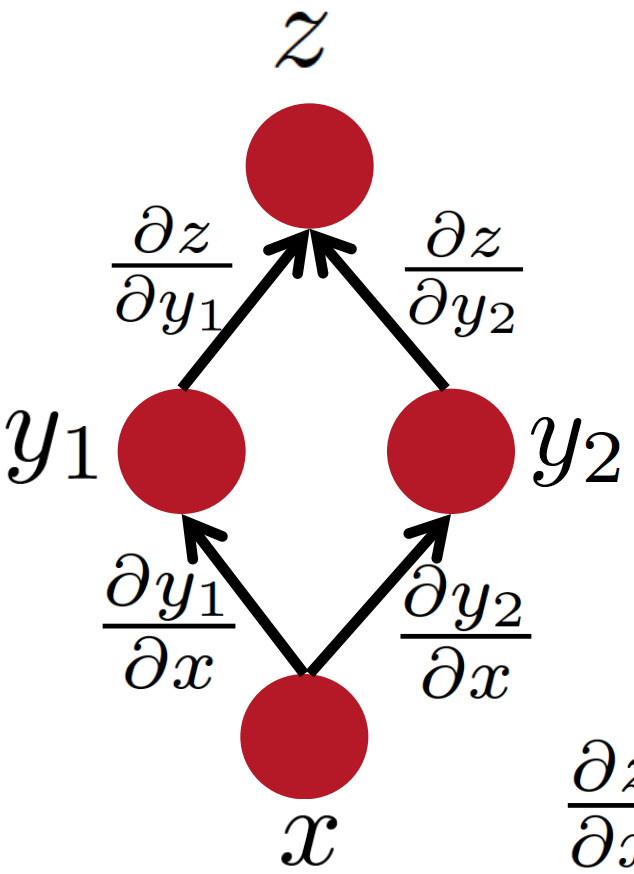
$$\Delta z = \frac{\partial z}{\partial y}\Delta y$$

$$\Delta y = \frac{\partial y}{\partial x}\Delta x$$

$$\Delta z = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}\Delta x$$

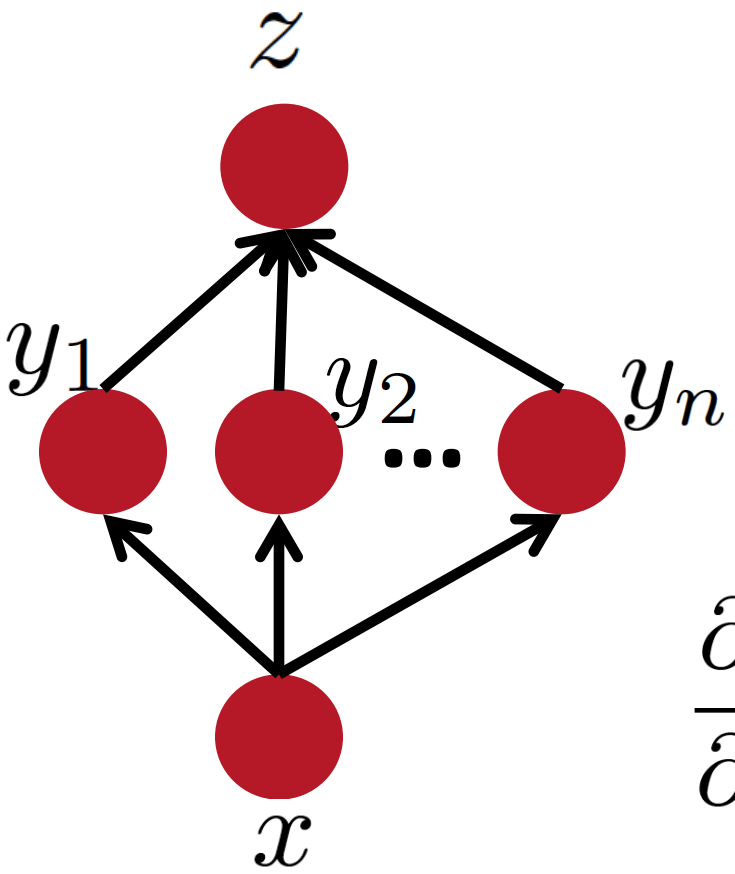
$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}$$

Multiple Paths Chain Rule



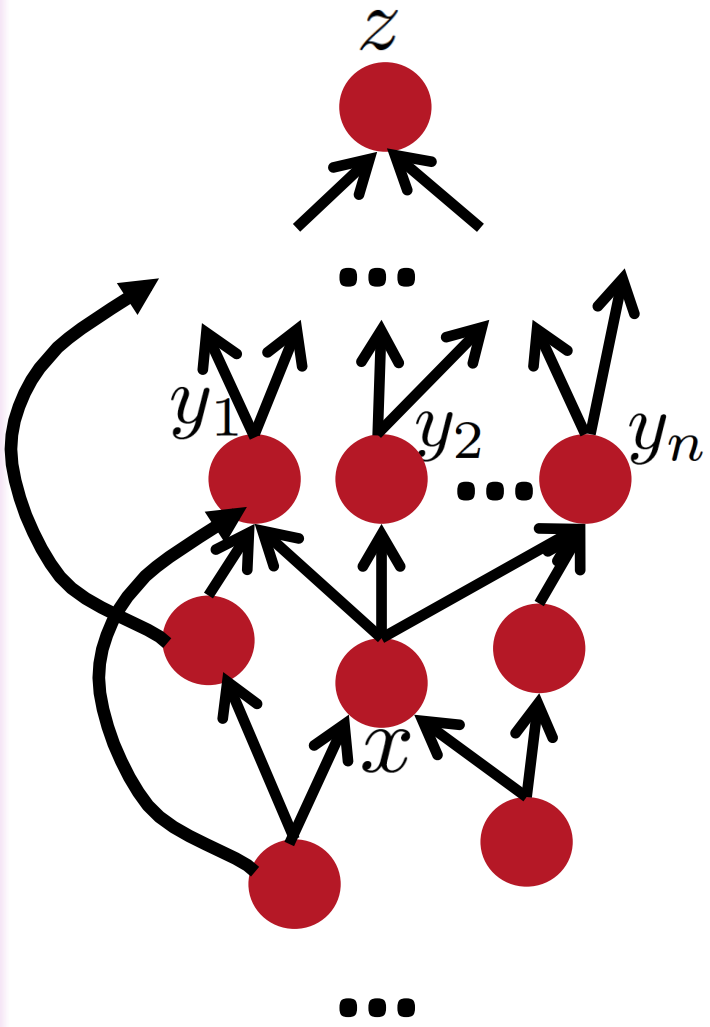
$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y_1}\frac{\partial y_1}{\partial x} + \frac{\partial z}{\partial y_2}\frac{\partial y_2}{\partial x}$$

Multiple Paths Chain Rule - General



$$\frac{\partial z}{\partial x} = \sum_{i=1}^{n} \frac{\partial z}{\partial y_i} \frac{\partial y_i}{\partial x}$$

Chain Rule in Flow Graph

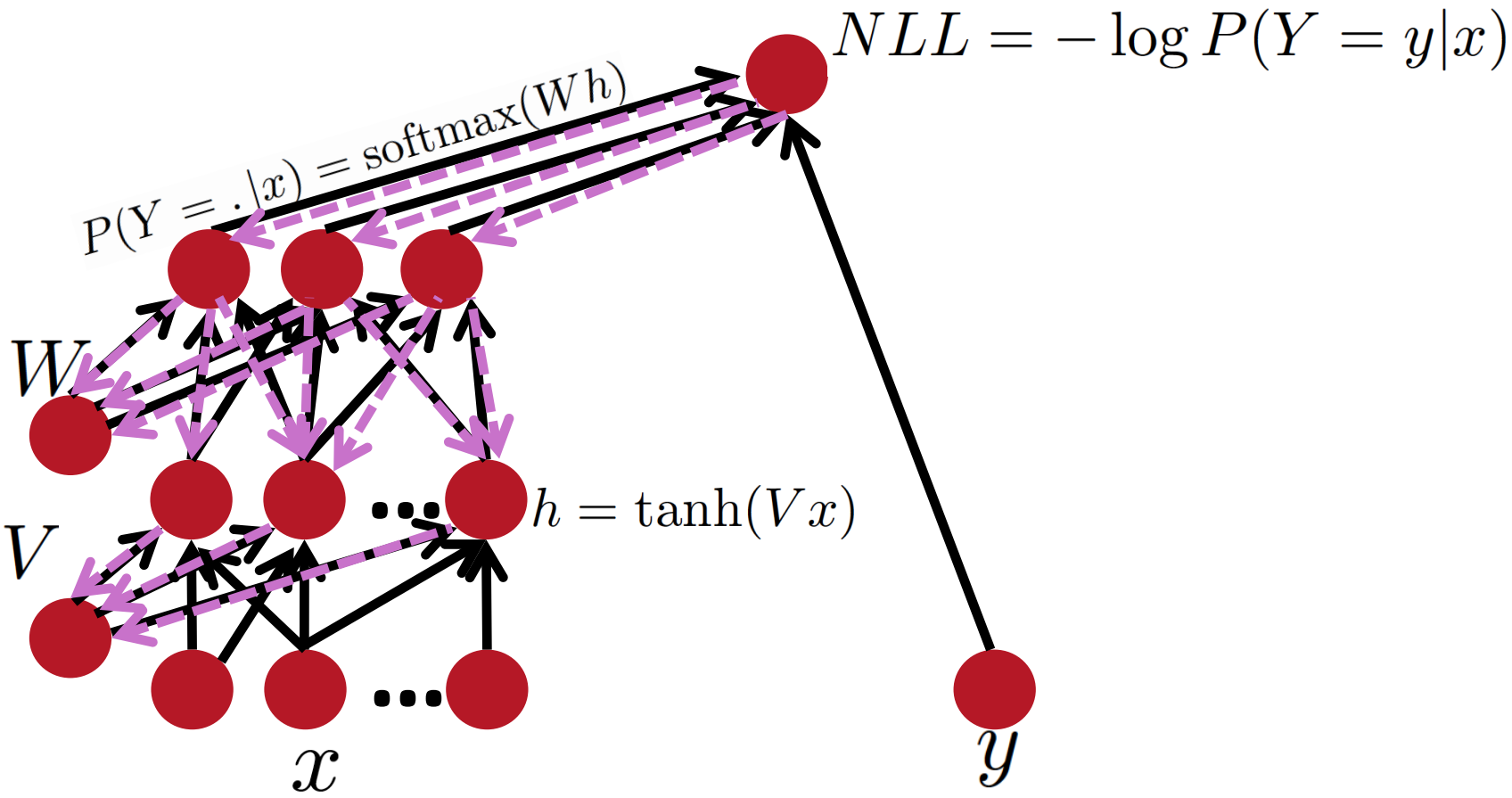


Flow graph: any directed acyclic graph
node = computation result
arc = computation dependency

$\{y_1, y_2, \ldots y_n\}$ = successors of x

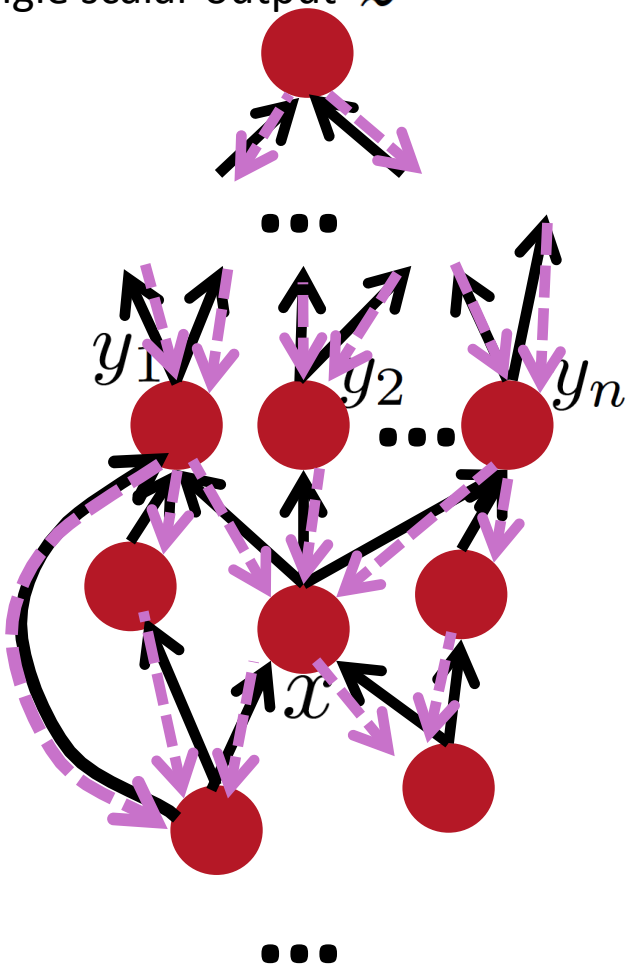
$$\frac{\partial z}{\partial x} = \sum_{i=1}^{n} \frac{\partial z}{\partial y_i} \frac{\partial y_i}{\partial x}$$

Back-Prop in Multi-Layer Net



Back-Prop in General Flow Graph

Single scalar output z



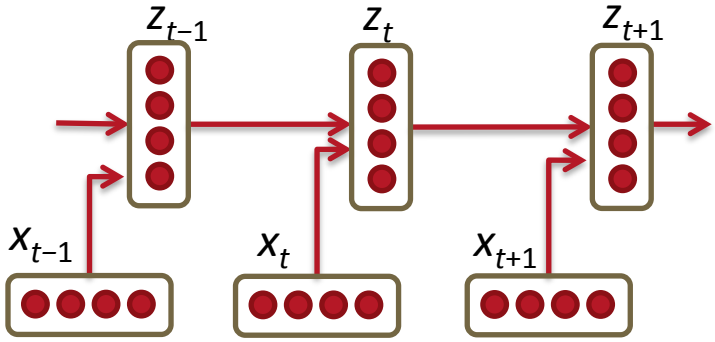
1. Fprop: visit nodes in topo-sort order
   - Compute value of node given predecessors
2. Bprop:
   - initialize output gradient = 1
   - visit nodes in reverse order:
     Compute gradient wrt each node using gradient wrt successors

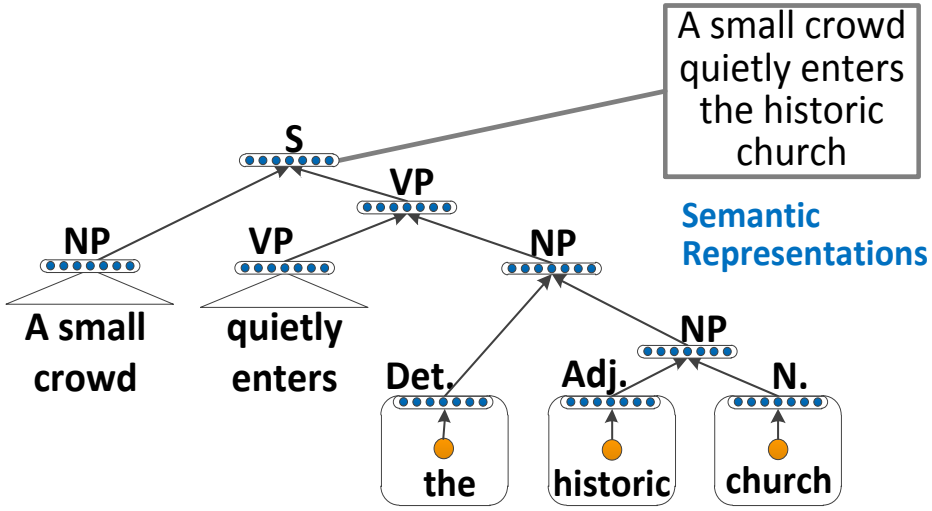
$\{y_1, y_2, \ldots y_n\}$ = successors of x

$$\frac{\partial z}{\partial x} = \sum_{i=1}^{n} \frac{\partial z}{\partial y_i} \frac{\partial y_i}{\partial x}$$

Back-Prop in Recurrent & Recursive Nets

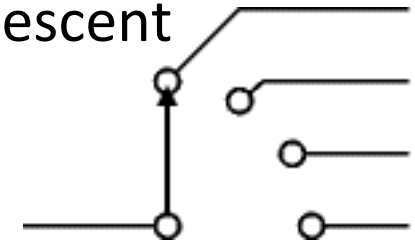
- Replicate a parameterized function over different time steps or nodes of a DAG
- Output state at one time-step / node is used as input for another time-step / node



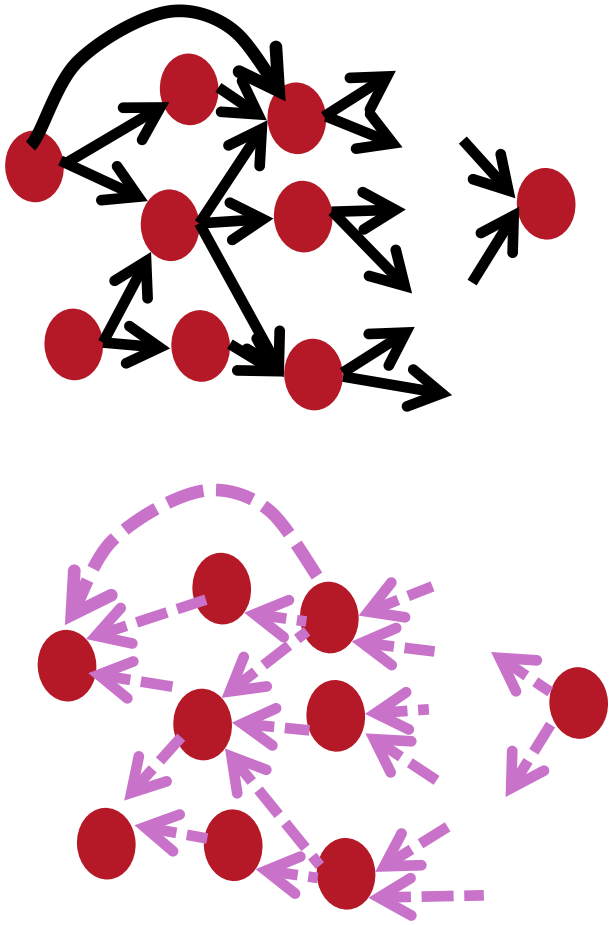


Backpropagation Through Structure

- Inference → discrete choices
  - (e.g., shortest path in HMM, best output configuration in CRF)
- E.g. Max over configurations or sum weighted by posterior
- The loss to be optimized depends on these choices
- The inference operations are flow graph nodes
- If continuous, can perform stochastic gradient descent
  - Max(a,b) is continuous.



Automatic Differentiation

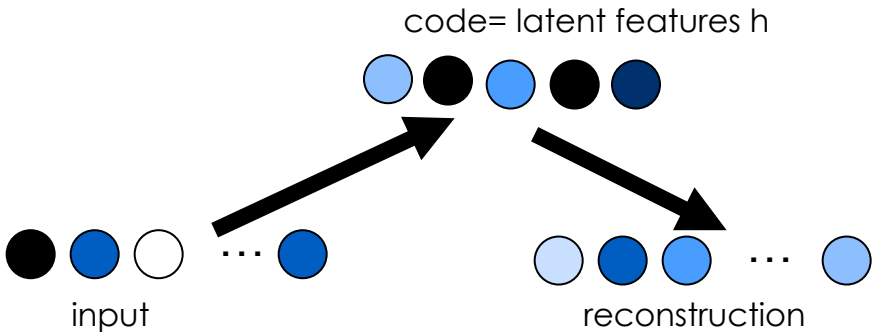


- The gradient computation can be automatically inferred from the symbolic expression of the fprop.
- Each node type needs to know how to compute its output and how to compute the gradient wrt its inputs given the gradient wrt its output.
- Easy and fast prototyping

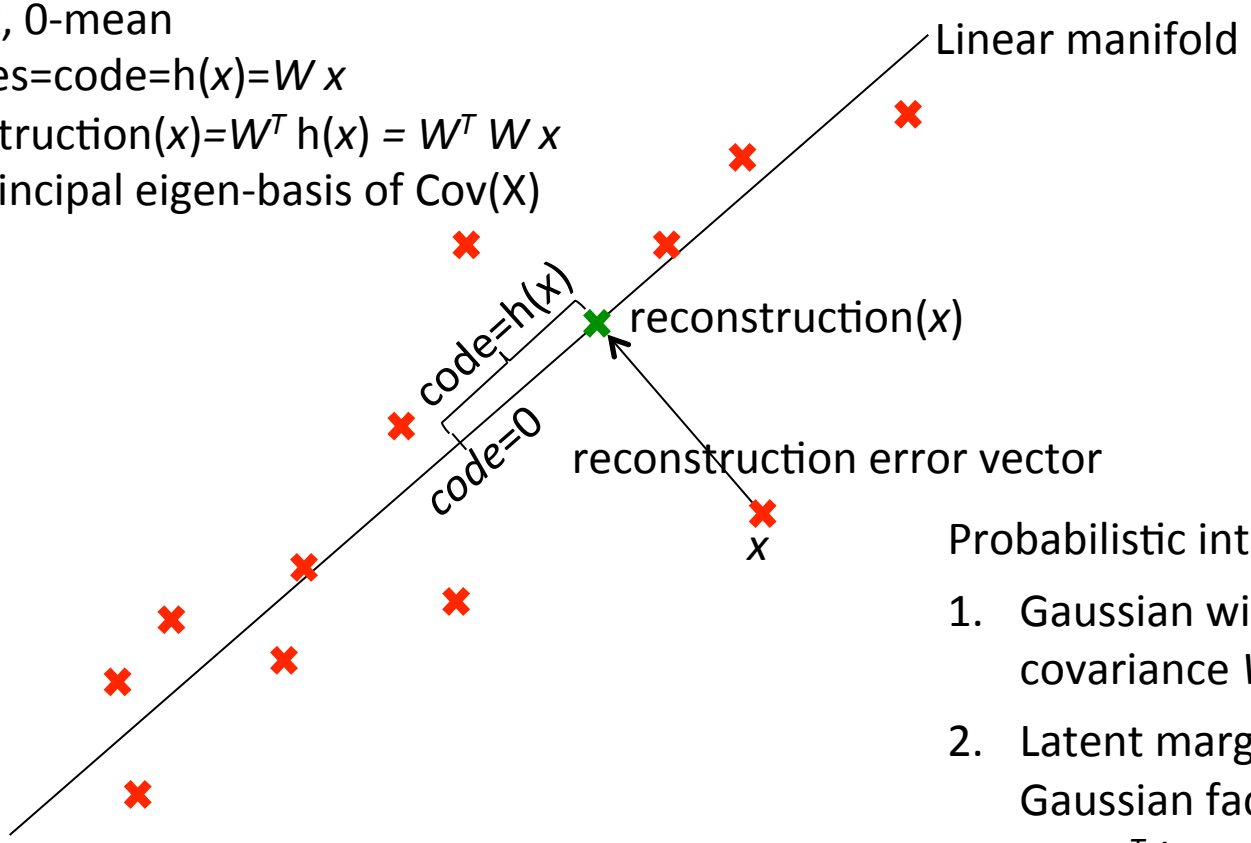
Distributed Representations and Neural Nets: How to do unsupervised training?

PCA

= Linear Manifold
= Linear Auto-Encoder
= Linear Gaussian Factors



input x, 0-mean
features=code=h(x)=$W\,x$
reconstruction(x)=W^T h(x) = $W^T\,W\,x$
W = principal eigen-basis of Cov(X)

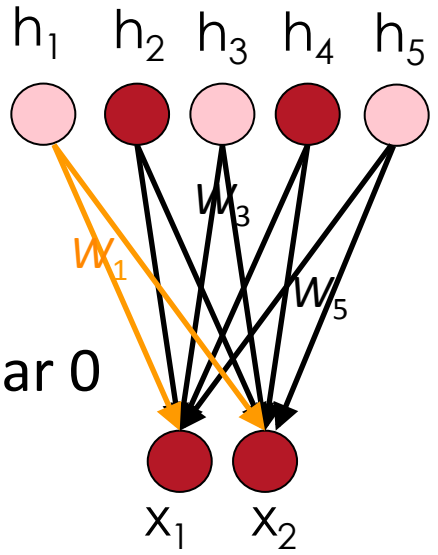


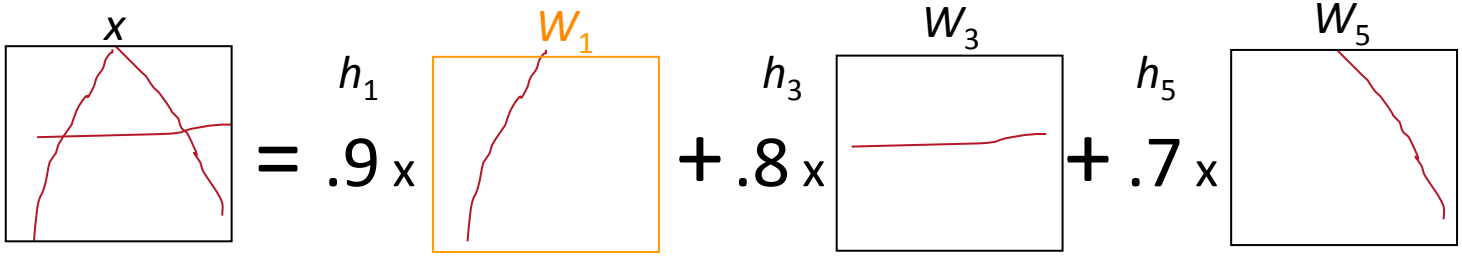
Probabilistic interpretations:

1. Gaussian with full covariance $W^T\,W+\lambda I$
2. Latent marginally iid Gaussian factors h with x = $W^T\,h$ + *noise*

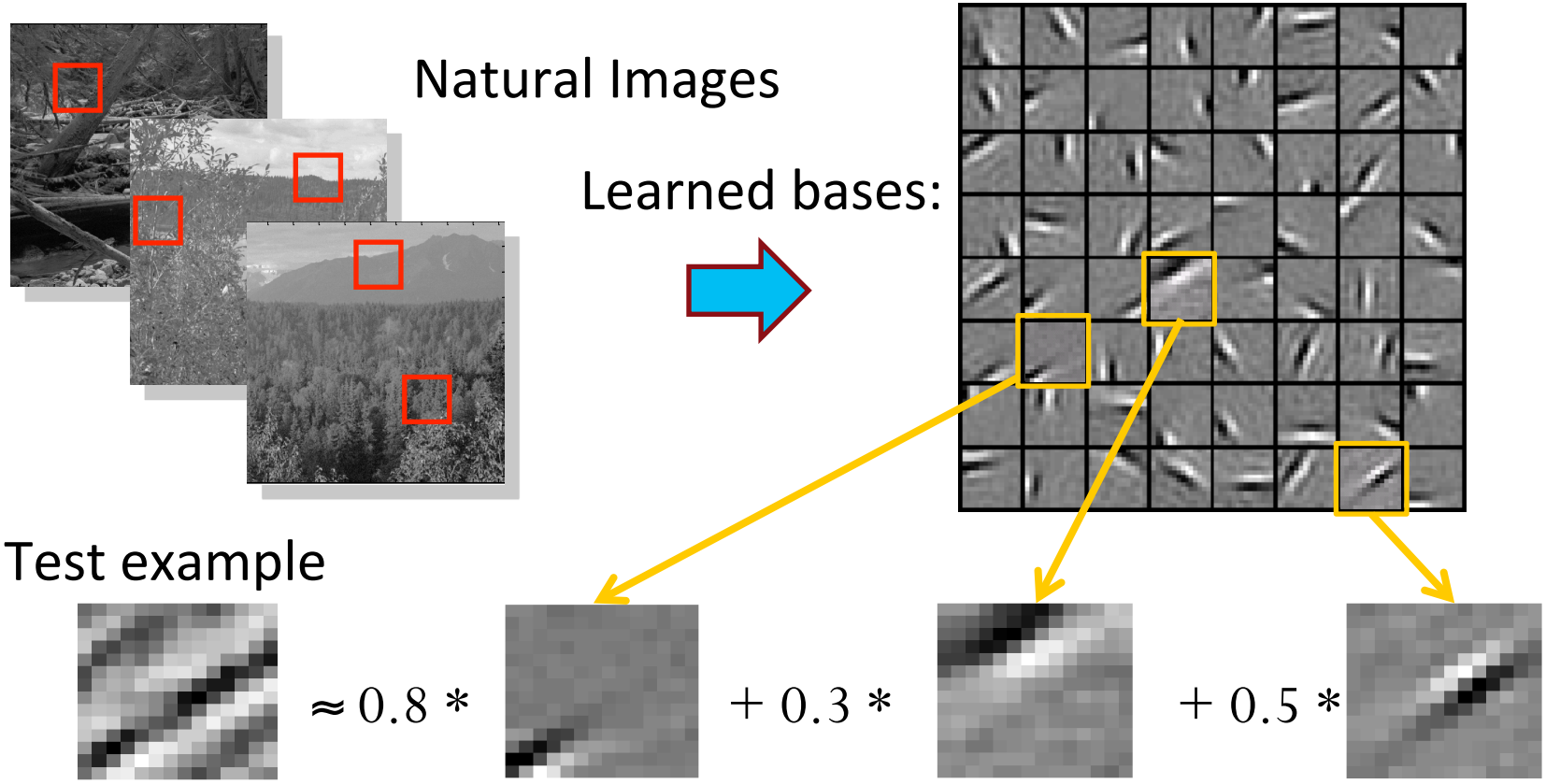
Directed Factor Models

- P(h) factorizes into P(h_1) P(h_2)…
- Different priors:
  - PCA: P(h_i) is Gaussian
  - ICA: P(h_i) is non-parametric
  - **Sparse coding**: P(h_i) is concentrated near 0
- Likelihood is typically Gaussian $x \mid h$ with mean given by $W^T h$
- Inference procedures (predicting h, given x) differ
- Sparse h: x is explained by the weighted addition of selected filters h_i





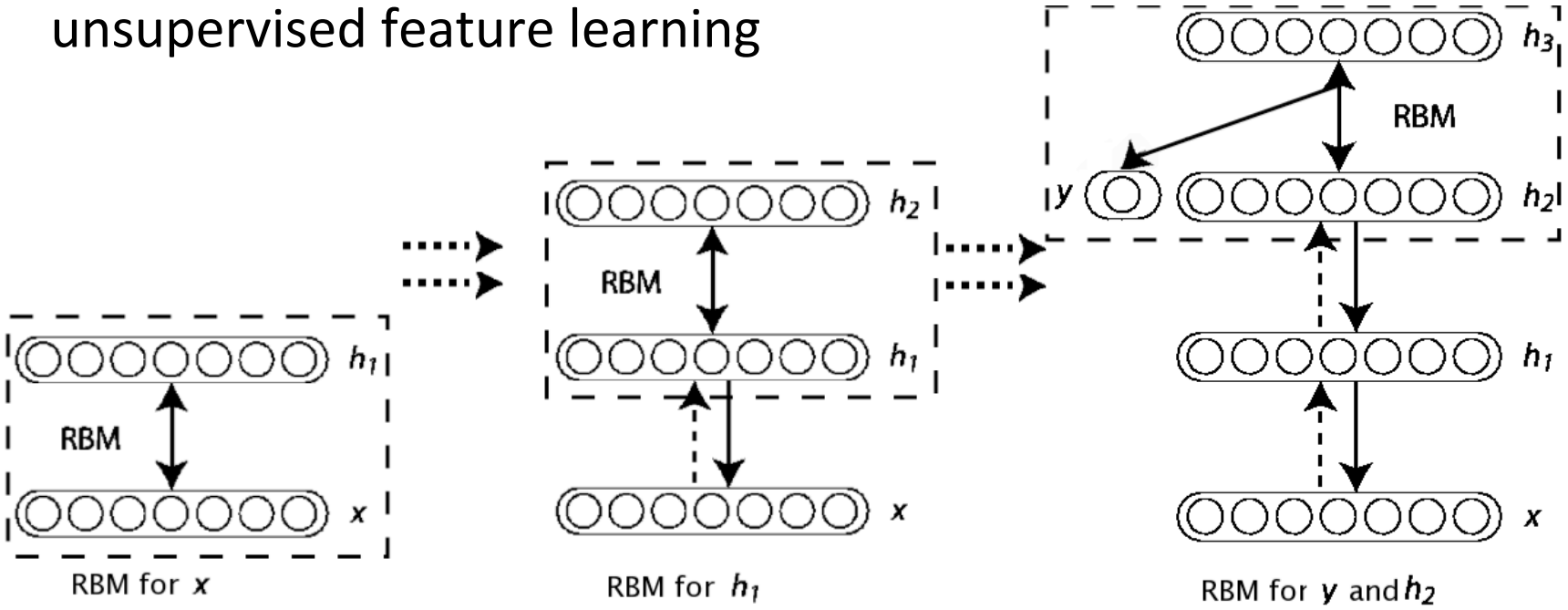
Sparse autoencoder illustration for images



$[a_1, \ldots, a_{64}] = [0, 0, \ldots, 0, \mathbf{0.8}, 0, \ldots, 0, \mathbf{0.3}, 0, \ldots, 0, \mathbf{0.5}, 0]$
(feature representation)

Stacking Single-Layer Learners

- PCA is great but can't be stacked into deeper more abstract representations (linear x linear = linear)
- One of the big ideas from Hinton et al. 2006: layer-wise unsupervised feature learning



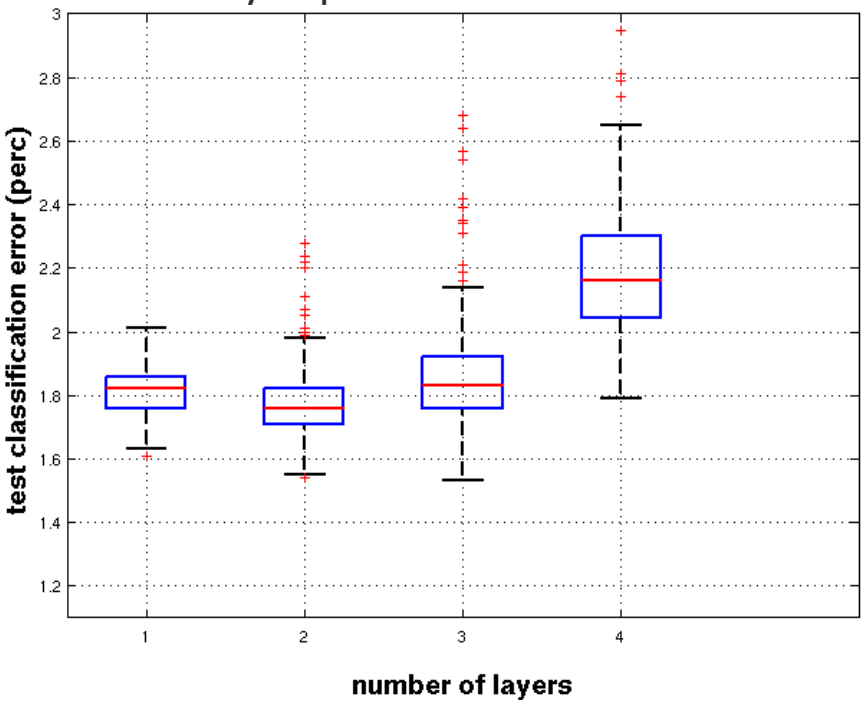
Stacking Restricted Boltzmann Machines (RBM) → Deep Belief Network (DBN)

Effective deep learning became possible through unsupervised pre-training

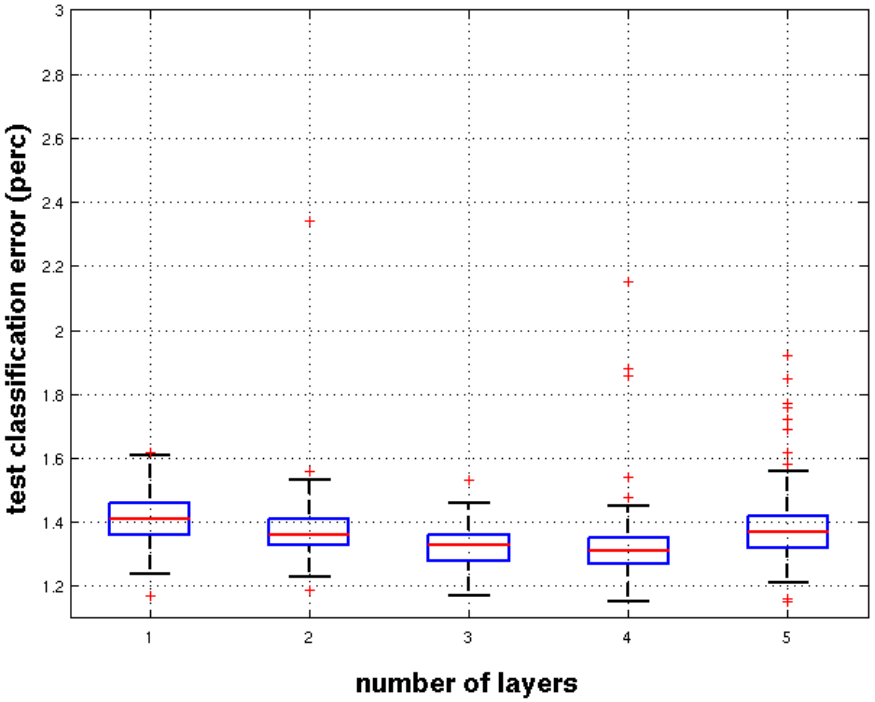
[Erhan et al., JMLR 2010]

(with RBMs and Denoising Auto-Encoders)

Purely supervised neural net



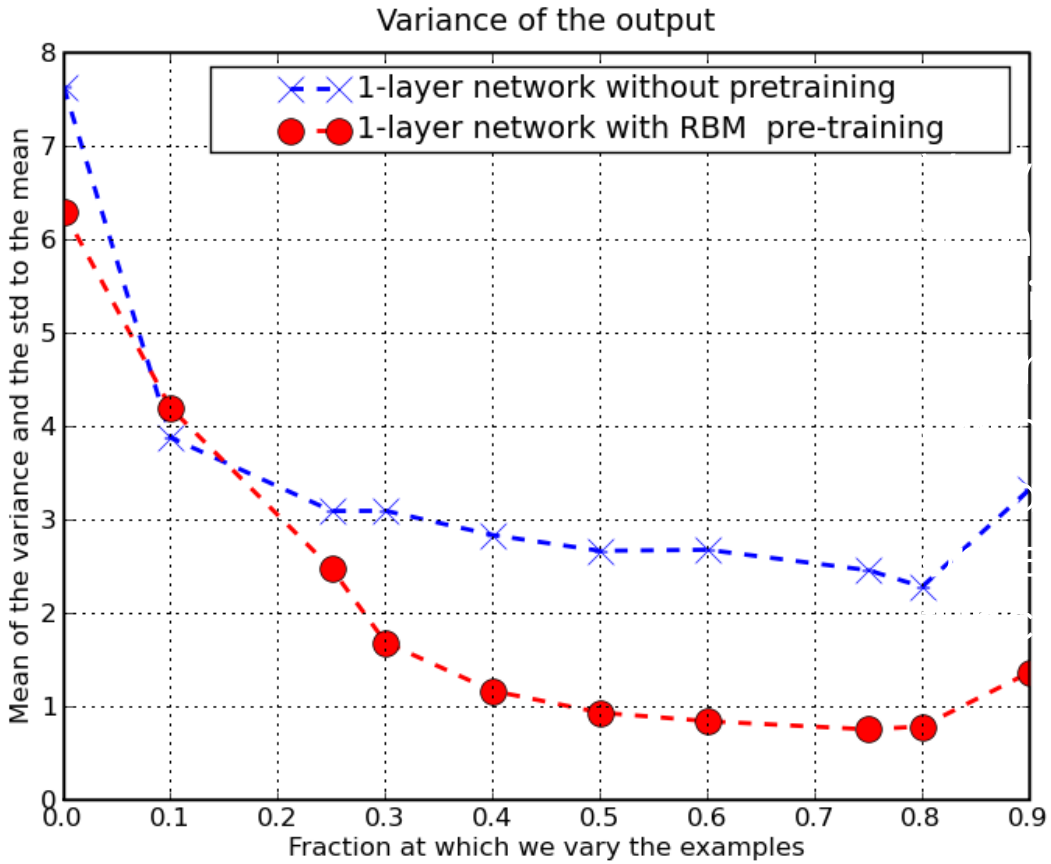
With unsupervised pre-training



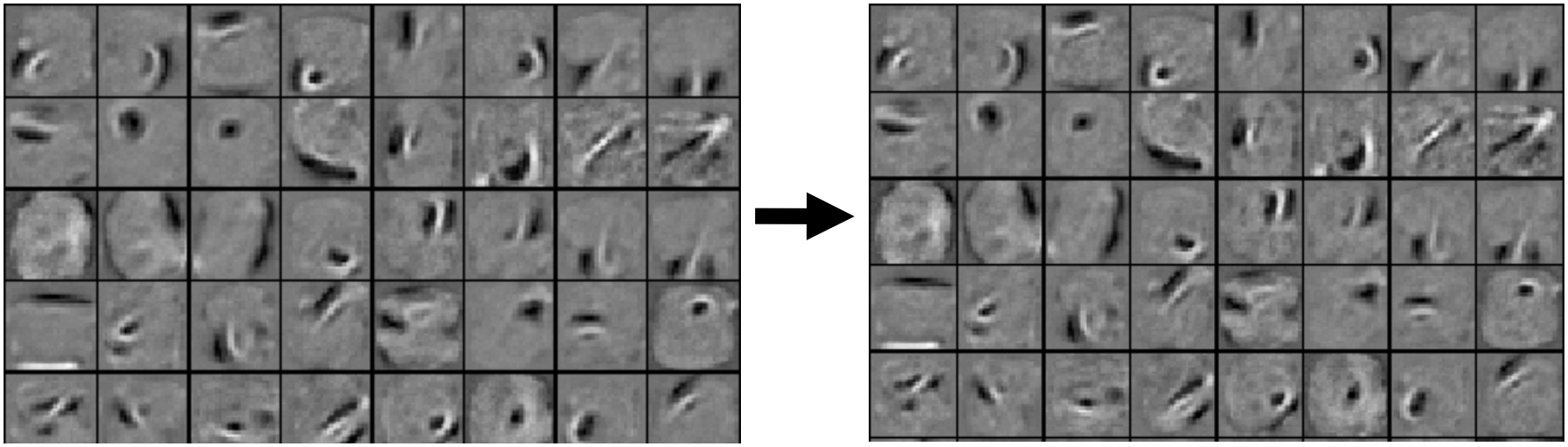
Optimizing Deep Non-Linear Composition of Functions Seems Hard

- Failure of training deep supervised nets before 2006
- Regularization effect vs optimization effect of unsupervised pre-training
- Is optimization difficulty due to
  - ill-conditioning?
  - local minima?
  - both?

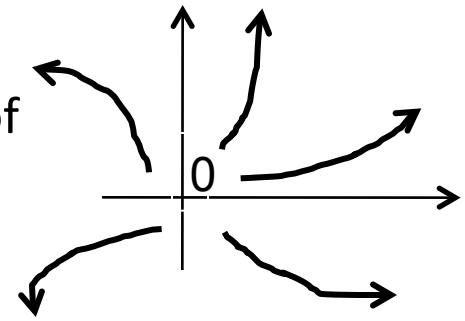
Initial Examples Matter More (critical period?)



Learning Dynamics of Deep Nets



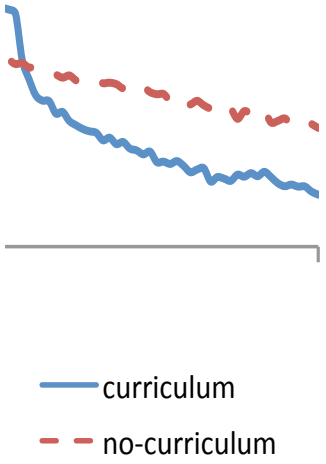
- As weights become larger, get trapped in basin of attraction (sign does not change)
- Critical period. Initialization matters.



Order & Selection of Examples Matters

(Bengio, Louradour, Collobert & Weston, ICML'2009)

- Curriculum learning
- (Bengio et al 2009, Krueger & Dayan 2009)
- Start with easier examples
- Faster convergence to a better local minimum in deep architectures
- Also acts like a regularizer with optimization effect?



Understanding the difficulty of training deep feedforward neural networks

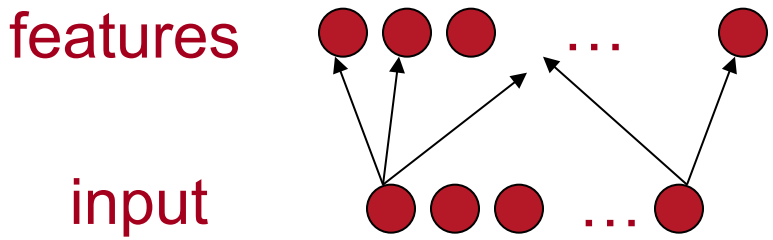
(Glorot & Bengio, AISTATS 2010)


Study the activations and gradients

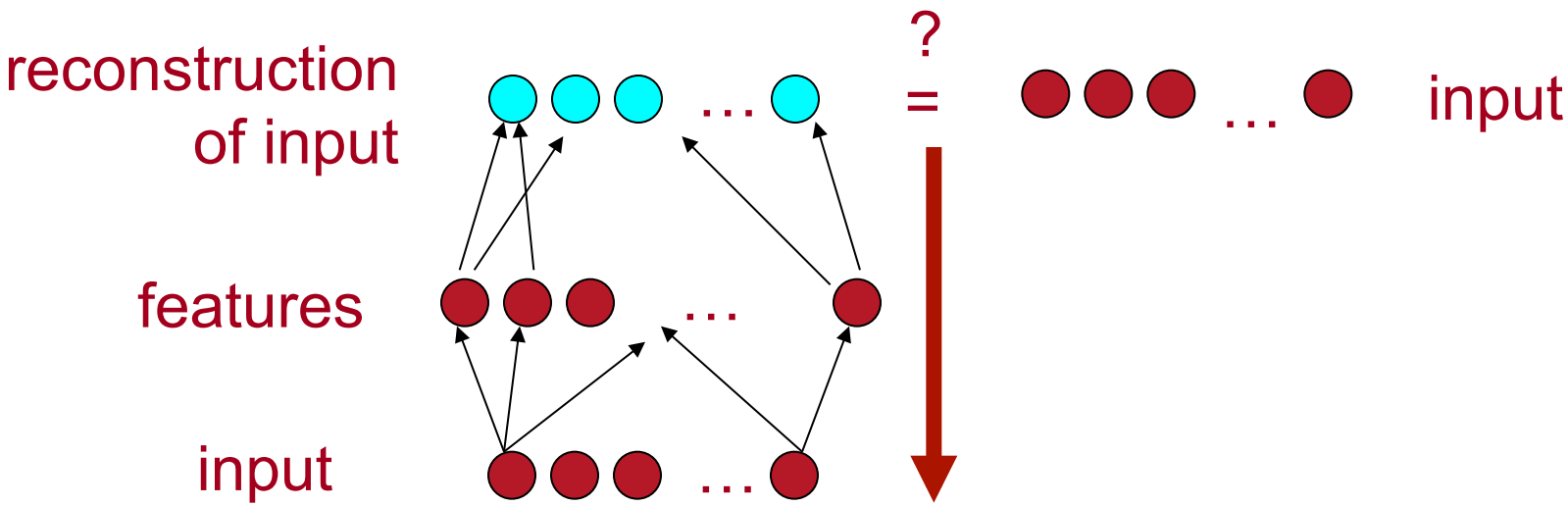
- wrt depth
- as training progresses
- for different initializations → big difference
- for different activation non-linearities

Layer-wise Unsupervised Learning

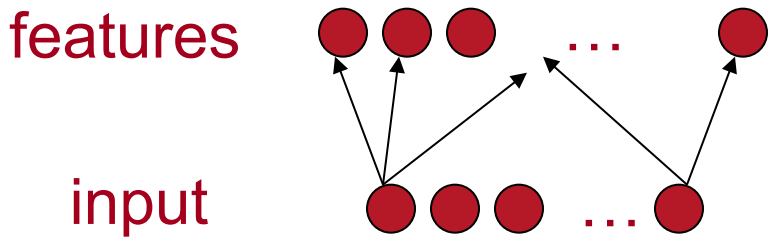
Layer-Wise Unsupervised Pre-training



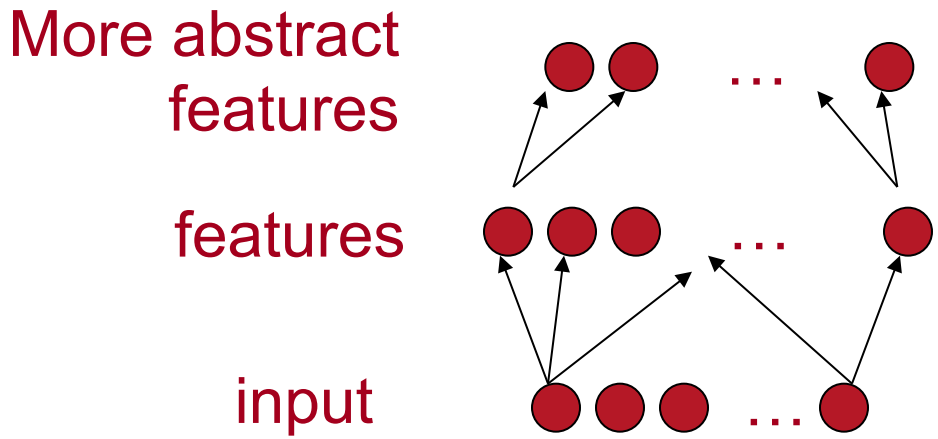
Layer-Wise Unsupervised Pre-training



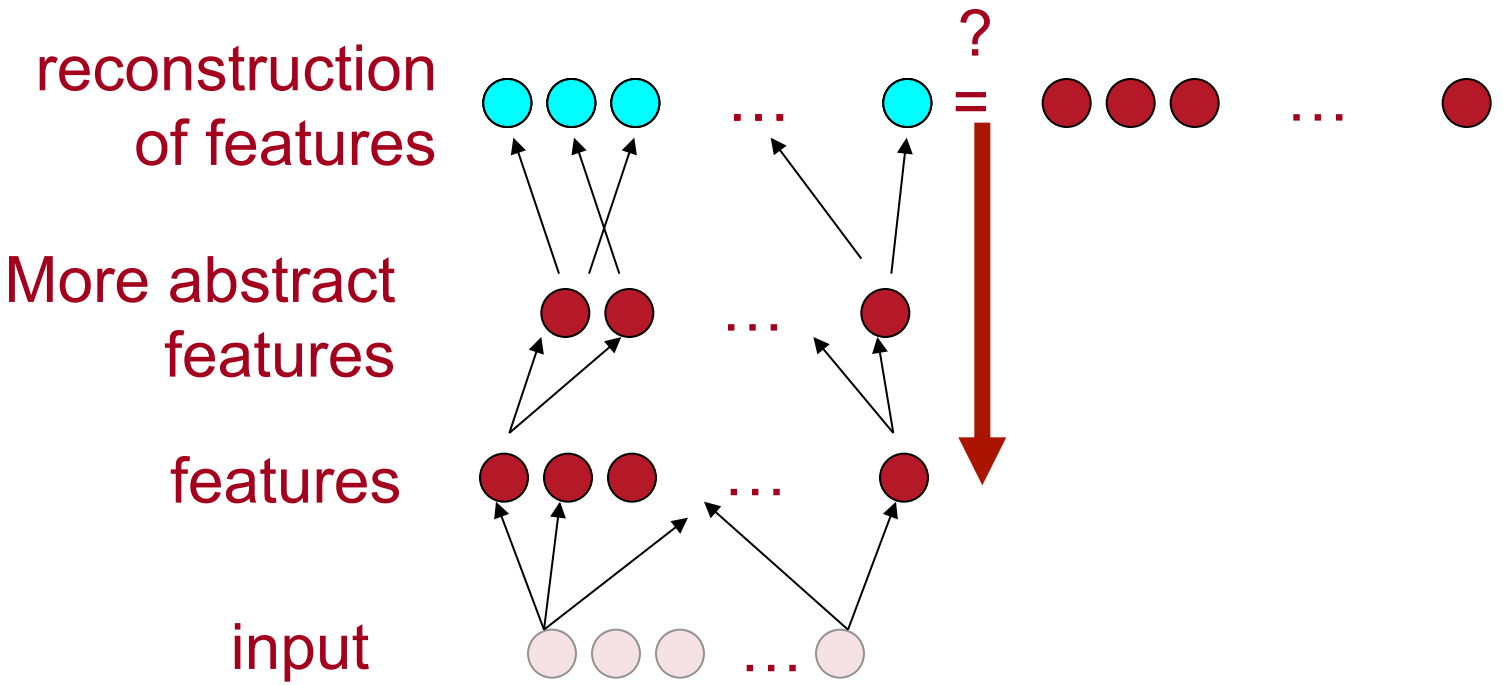
Layer-Wise Unsupervised Pre-training



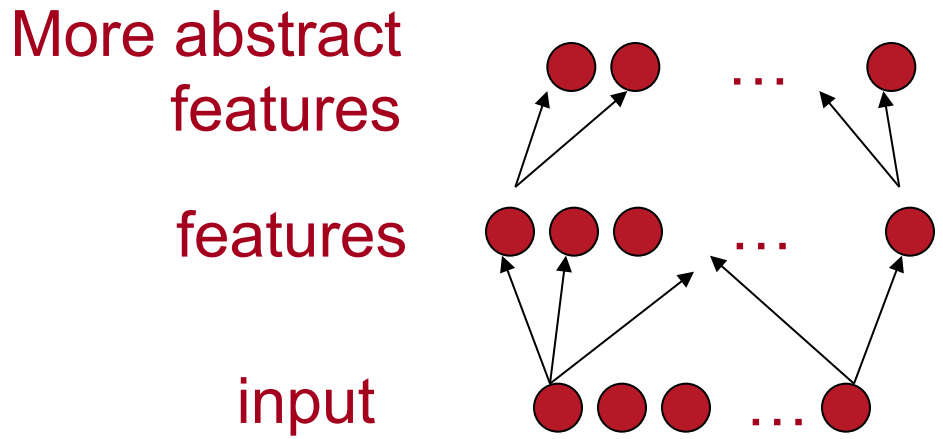
Layer-Wise Unsupervised Pre-training



Layer-wise Unsupervised Learning



Layer-Wise Unsupervised Pre-training



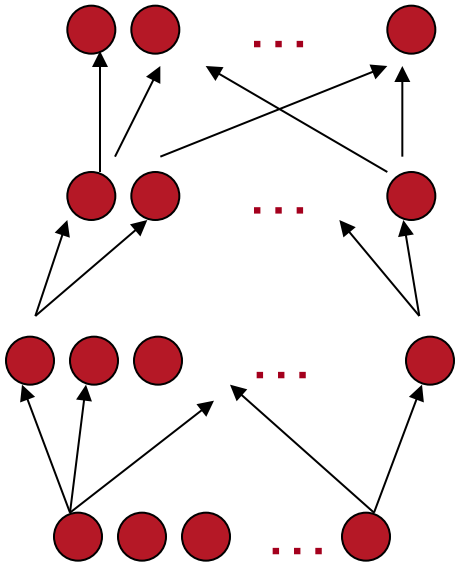
Layer-wise Unsupervised Learning

Even more abstract features

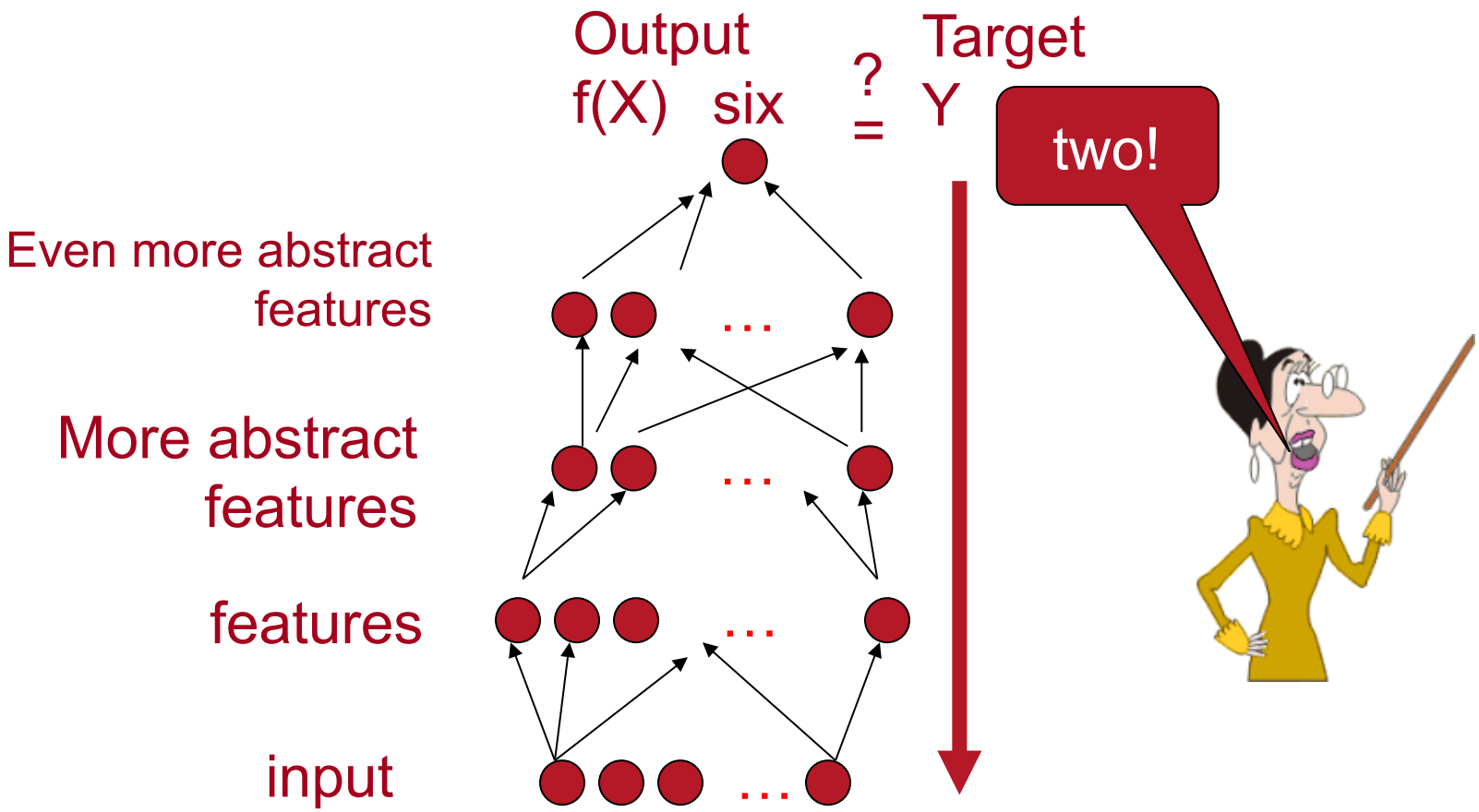
More abstract features

features

input



Supervised Fine-Tuning

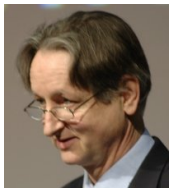


- Additional hypothesis: features good for P(x) good for P(y|x)

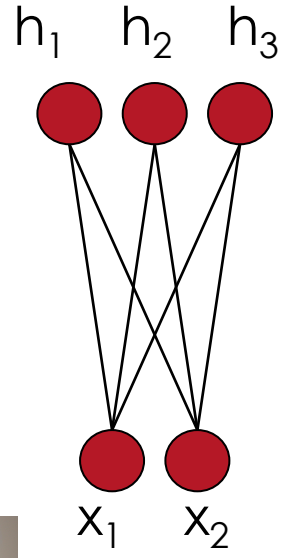
Restricted Boltzmann Machines

Undirected Models: the Restricted Boltzmann Machine

[Hinton et al 2006]



- Probabilistic model of the joint distribution of the observed variables (inputs alone or inputs and targets) *x*
- Latent (hidden) variables *h* model high-order dependencies
- Inference is easy, P(*h*|*x*) factorizes



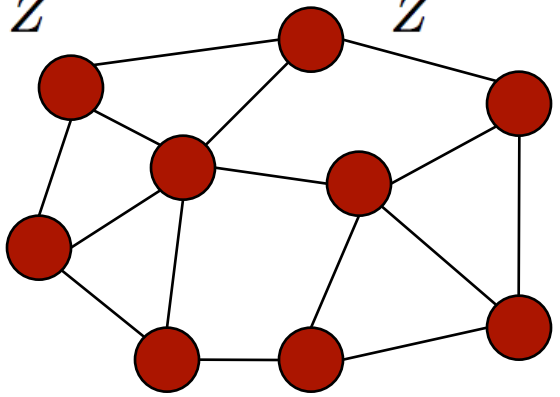
- See Bengio (2009) detailed monograph/review: *"Learning Deep Architectures for AI"*.
- See Hinton (2010) *"A practical guide to training Restricted Boltzmann Machines"*

Boltzmann Machines & MRFs

- Boltzmann machines:

(Hinton 84) $$P(x) = \frac{1}{Z} e^{-\text{Energy}(x)} = \frac{1}{Z} e^{c^T x + x^T W x} = \frac{1}{Z} e^{\sum_i c_i x_i + \sum_{i,j} x_i W_{ij} x_j}$$

- Markov Random Fields:



Undirected graphical models

$$P(x) = \frac{1}{Z} e^{\sum_i w_i f_i(x)}$$

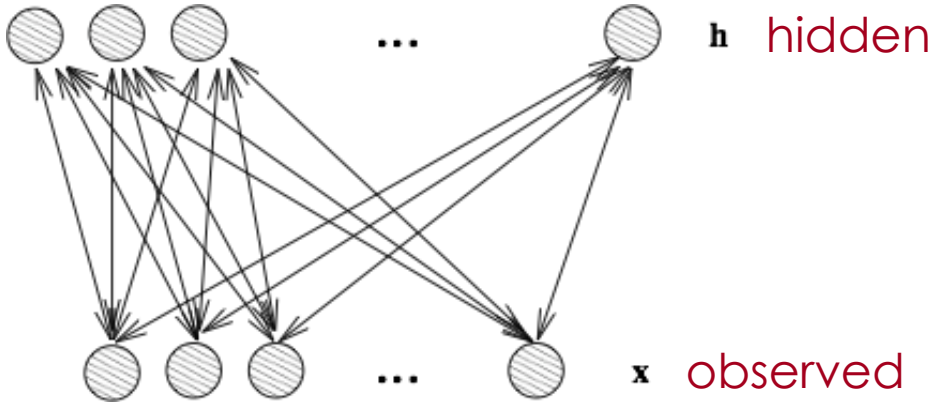
Soft constraint / probabilistic statement

- More interesting with latent variables!

Restricted Boltzmann Machine (RBM)

$$P(x,h) = \frac{1}{Z} e^{b^T h + c^T x + h^T W x} = \frac{1}{Z} e^{\sum_i b_i h_i + \sum_j c_j x_j + \sum_{i,j} h_i W_{ij} x_j}$$

- A popular building block for deep architectures
- **Bipartite** undirected graphical model



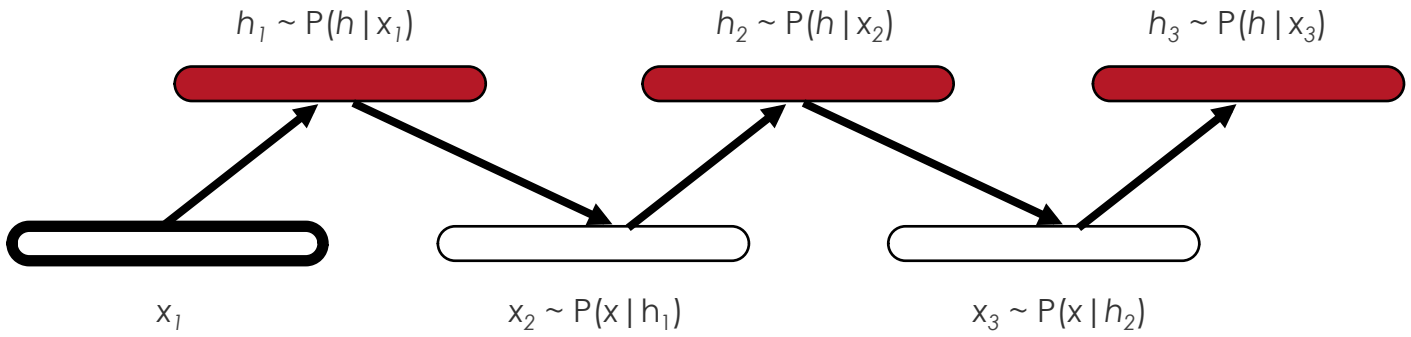
Gibbs Sampling & Block Gibbs Sampling

- Want to sample from $P(X_1,X_2,\ldots X_n)$
- **Gibbs sampling**
  - Iterate or randomly choose i in {1…n}
    - Sample X_i from $P(X_i \mid X_1,X_2,\ldots X_{i-1}, X_{i+1},\ldots X_n)$

  can only make small changes at a time! → slow mixing
  Note how fixed point samples from the joint.

- **Block Gibbs sampling**
  - X's organized in blocks, e.g. $A=(X_1,X_2,X_3)$, $B=(X_4,X_5,X_6)$, C=…
  - Do Gibbs on P(A,B,C,…), i.e.
    - Sample A from P(A|B,C)
    - Sample B from P(B|A,C)
    - Sample C from P(C|A,B), and iterate…
  - Larger changes → faster mixing

Gibbs Sampling in RBMs



$P(h|x)$ and $P(x|h)$ factorize

$P(h|x) = \prod_i P(h_i|x)$

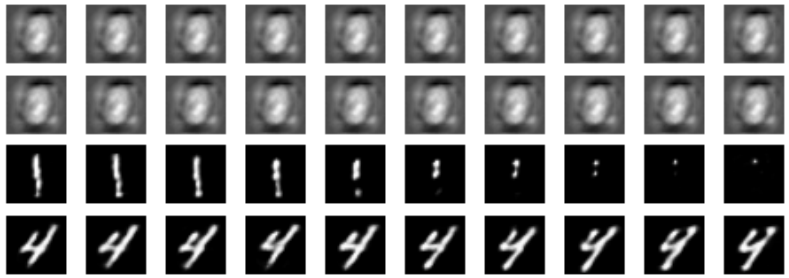
- Easy inference
- Efficient **block Gibbs** sampling x→h→x→h…

$$P(x,h) = \frac{1}{Z} e^{b^T h + c^T x + h^T W x}$$

Problems with Gibbs Sampling

In practice, Gibbs sampling does not always mix well…

RBM trained by CD on MNIST

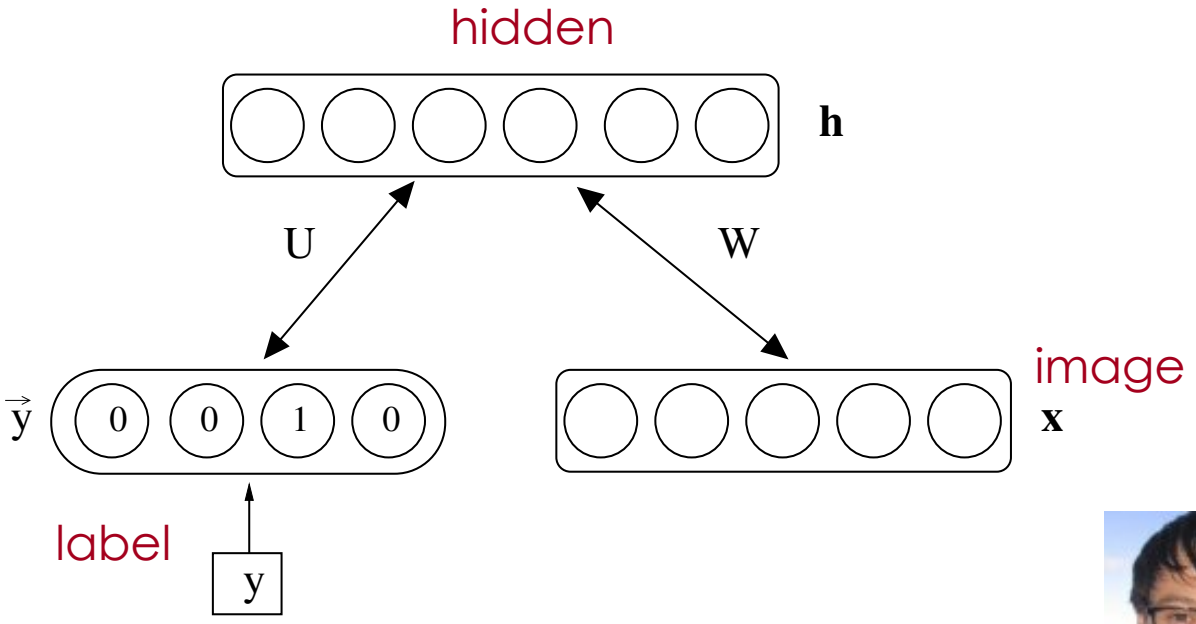


Chains from random state

Chains from real digits

(Desjardins et al 2010)

RBM with (image, label) visible units



(Larochelle & Bengio 2008)

RBMs are Universal Approximators

(Le Roux & Bengio 2008)


- Adding one hidden unit (with proper choice of parameters) guarantees increasing likelihood
- With enough hidden units, can perfectly model any discrete distribution
- RBMs with variable # of hidden units = non-parametric

RBM Conditionals Factorize

$$
\begin{aligned}
P(\mathbf{h}|\mathbf{x}) &= \frac{\exp(\mathbf{b}'\mathbf{x} + \mathbf{c}'\mathbf{h} + \mathbf{h}'W\mathbf{x})}{\sum_{\tilde{\mathbf{h}}} \exp(\mathbf{b}'\mathbf{x} + \mathbf{c}'\tilde{\mathbf{h}} + \tilde{\mathbf{h}}'W\mathbf{x})} \\
&= \frac{\prod_i \exp(\mathbf{c}_i\mathbf{h}_i + \mathbf{h}_i W_i\mathbf{x})}{\prod_i \sum_{\tilde{\mathbf{h}}_i} \exp(\mathbf{c}_i\tilde{\mathbf{h}}_i + \tilde{\mathbf{h}}_i W_i\mathbf{x})} \\
&= \prod_i \frac{\exp(\mathbf{h}_i(\mathbf{c}_i + W_i\mathbf{x}))}{\sum_{\tilde{\mathbf{h}}_i} \exp(\tilde{\mathbf{h}}_i(\mathbf{c}_i + W_i\mathbf{x}))} \\
&= \prod_i P(\mathbf{h}_i|\mathbf{x}).
\end{aligned}
$$

RBM Energy Gives Binomial Neurons

With $\mathbf{h}_i \in \{0, 1\}$, recall $\text{Energy}(\mathbf{x}, \mathbf{h}) = -\mathbf{b}'\mathbf{x} - \mathbf{c}'\mathbf{h} - \mathbf{h}'W\mathbf{x}$

$$
\begin{aligned}
P(\mathbf{h}_i = 1|\mathbf{x}) &= \frac{e^{1\mathbf{c}_i + 1W_i\mathbf{x} + other\ terms}}{e^{1\mathbf{c}_i + 1W_i\mathbf{x} + other\ terms} + e^{0\mathbf{c}_i + 0W_i\mathbf{x} + other\ terms}} \\
&= \frac{e^{\mathbf{c}_i + W_i\mathbf{x}}}{e^{\mathbf{c}_i + W_i\mathbf{x}} + 1} \\
&= \frac{1}{1 + e^{-\mathbf{c}_i - W_i\mathbf{x}}} \\
&= \text{sigm}(\mathbf{c}_i + W_i\mathbf{x}).
\end{aligned}
$$

since $\text{sigm}(a) = \frac{1}{1+e^{-a}}$.

RBM Free Energy

$$P(\mathbf{x}, \mathbf{h}) = \frac{e^{-\text{Energy}(\mathbf{x},\mathbf{h})}}{Z}$$

- Free Energy = equivalent energy when marginalizing

$$P(\mathbf{x}) = \sum_{\mathbf{h}} \frac{e^{-\text{Energy}(\mathbf{x},\mathbf{h})}}{Z} = \frac{e^{-\text{FreeEnergy}(\mathbf{x})}}{Z}$$

- Can be computed exactly and efficiently in RBMs

$$\text{FreeEnergy}(\mathbf{x}) = -\mathbf{b}'\mathbf{x} - \sum_i \log \sum_{\mathbf{h}_i} e^{\mathbf{h}_i(\mathbf{c}_i + W_i\mathbf{x})}$$

- Marginal likelihood *P(**x**)* tractable up to partition function *Z*

Factorization of the Free Energy

Let the energy have the following general form:

$$\text{Energy}(\mathbf{x}, \mathbf{h}) = -\beta(\mathbf{x}) + \sum_i \gamma_i(\mathbf{x}, \mathbf{h}_i)$$

Then

$$\begin{aligned}
P(\mathbf{x}) &= \frac{1}{Z} e^{-\text{FreeEnergy}(\mathbf{x})} = \frac{1}{Z} \sum_{\mathbf{h}} e^{-\text{Energy}(\mathbf{x},\mathbf{h})} \\
&= \frac{1}{Z} \sum_{\mathbf{h}_1} \sum_{\mathbf{h}_2} \cdots \sum_{\mathbf{h}_k} e^{\beta(\mathbf{x}) - \sum_i \gamma_i(\mathbf{x},\mathbf{h}_i)} = \frac{1}{Z} \sum_{\mathbf{h}_1} \sum_{\mathbf{h}_2} \cdots \sum_{\mathbf{h}_k} e^{\beta(\mathbf{x})} \prod_i e^{-\gamma_i(\mathbf{x},\mathbf{h}_i)} \\
&= \frac{e^{\beta(\mathbf{x})}}{Z} \sum_{\mathbf{h}_1} e^{-\gamma_1(\mathbf{x},\mathbf{h}_1)} \sum_{\mathbf{h}_2} e^{-\gamma_2(\mathbf{x},\mathbf{h}_2)} \cdots \sum_{\mathbf{h}_k} e^{-\gamma_k(\mathbf{x},\mathbf{h}_k)} \\
&= \frac{e^{\beta(\mathbf{x})}}{Z} \prod_i \sum_{\mathbf{h}_i} e^{-\gamma_i(\mathbf{x},\mathbf{h}_i)}
\end{aligned}$$

$$\text{FreeEnergy}(\mathbf{x}) = -\log P(\mathbf{x}) - \log Z = -\beta(\mathbf{x}) - \sum_i \log \sum_{\mathbf{h}_i} e^{-\gamma_i(\mathbf{x},\mathbf{h}_i)}$$

Energy-Based Models Gradient

$$P(\mathbf{x}) = \frac{e^{-\text{Energy}(\mathbf{x})}}{Z} \qquad Z = \sum_{\mathbf{x}} e^{-\text{Energy}(\mathbf{x})}$$

$$\frac{\partial \log P(\mathbf{x})}{\partial \theta} = -\frac{\partial \text{Energy}(\mathbf{x})}{\partial \theta} - \frac{\partial \log Z}{\partial \theta}$$

$$\begin{aligned}
\frac{\partial \log Z}{\partial \theta} &= \frac{\partial \log \sum_{\mathbf{x}} e^{-\text{Energy}(\mathbf{x})}}{\partial \theta} \\
&= \frac{1}{Z} \frac{\partial \sum_{\mathbf{x}} e^{-\text{Energy}(\mathbf{x})}}{\partial \theta} \\
&= -\frac{1}{Z} \sum_{\mathbf{x}} e^{-\text{Energy}(\mathbf{x})} \frac{\partial \text{Energy}(\mathbf{x})}{\partial \theta} \\
&= -\sum_{\mathbf{x}} P(\mathbf{x}) \frac{\partial \text{Energy}(\mathbf{x})}{\partial \theta}
\end{aligned}$$

Boltzmann Machine Gradient

$$P(x) = \frac{1}{Z}\sum_h e^{-\text{Energy}(x,h)} = \frac{1}{Z}e^{-\text{FreeEnergy}(x)}$$

- Gradient has two components:

"positive phase" "negative phase"

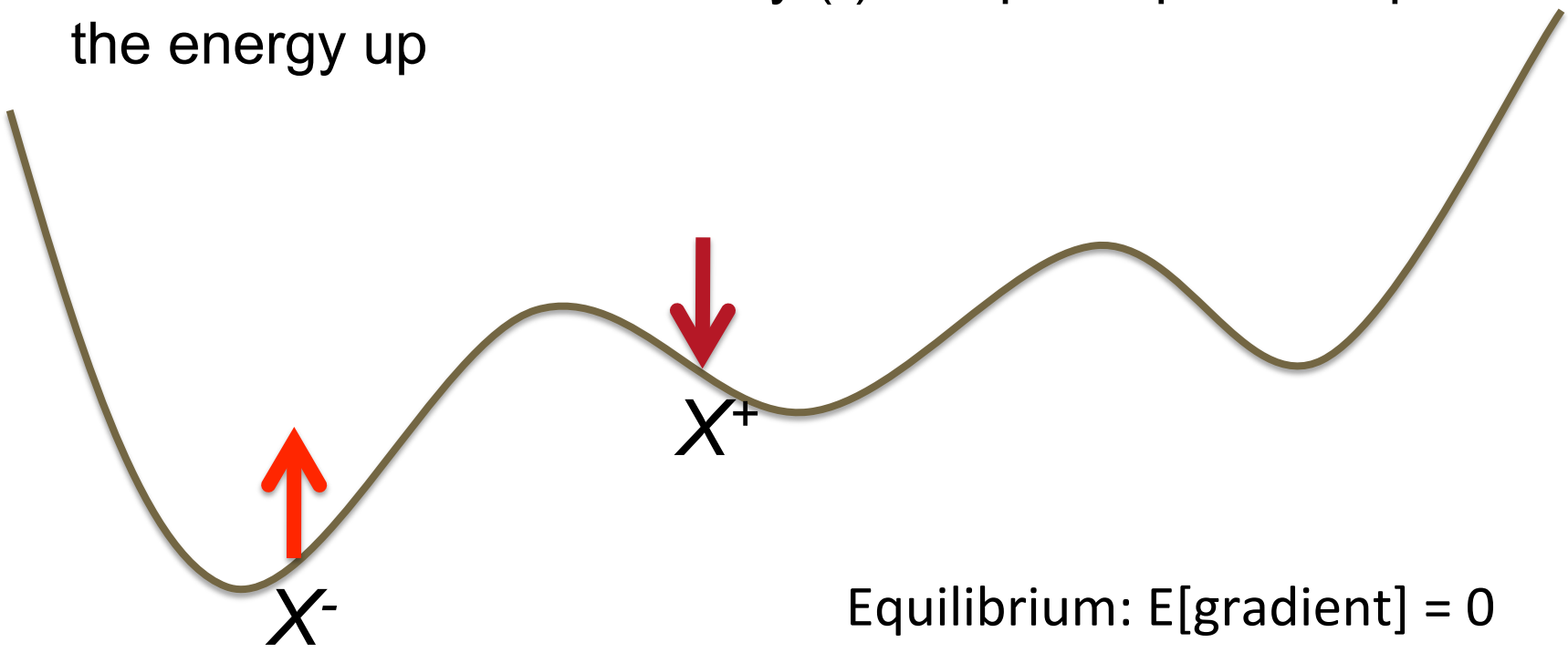
$$\frac{\partial \log P(x)}{\partial \theta} = -\frac{\partial \text{FreeEnergy}(x)}{\partial \theta} + \sum_{\tilde{x}} P(\tilde{x}) \frac{\partial \text{FreeEnergy}(\tilde{x})}{\partial \theta}$$

$$= -\sum_h P(h|x) \frac{\partial \text{Energy}(x,h)}{\partial \theta} + \sum_{\tilde{x},\tilde{h}} P(\tilde{x},\tilde{h}) \frac{\partial \text{Energy}(\tilde{x},\tilde{h})}{\partial \theta}$$

- In RBMs, easy to sample or sum over $h|x$
- Difficult part: sampling from $P(x)$, typically with a Markov chain

Positive & Negative Samples

- Observed (+) examples push the energy down
- Generated / dream / fantasy (-) samples / particles push the energy up

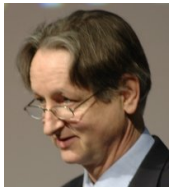


Equilibrium: E[gradient] = 0

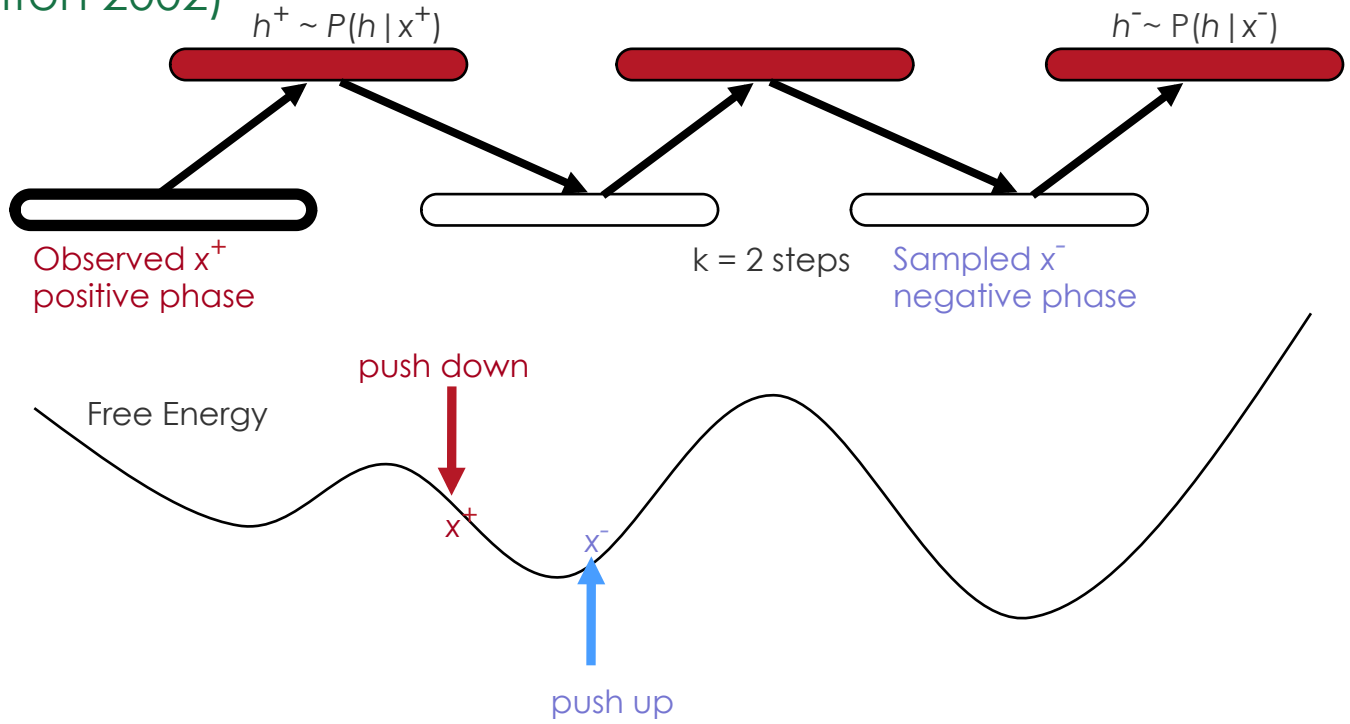
Training RBMs

Contrastive Divergence: (CD-k)	start negative Gibbs chain at observed x, run k Gibbs steps
SML/Persistent CD: (PCD)	run negative Gibbs chain in background while weights slowly change
Fast PCD:	two sets of weights, one with a large learning rate only used for negative phase, quickly exploring modes
Herding:	Deterministic near-chaos dynamical system defines both learning and sampling
Tempered MCMC:	use higher temperature to escape modes

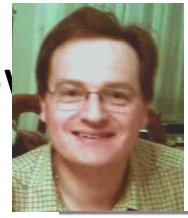
Contrastive Divergence



Contrastive Divergence (CD-k): start negative phase block Gibbs chain at observed x, run k Gibbs steps (Hinton 2002)



Persistent CD (PCD) / Stochastic Max. Likelihood (SML)

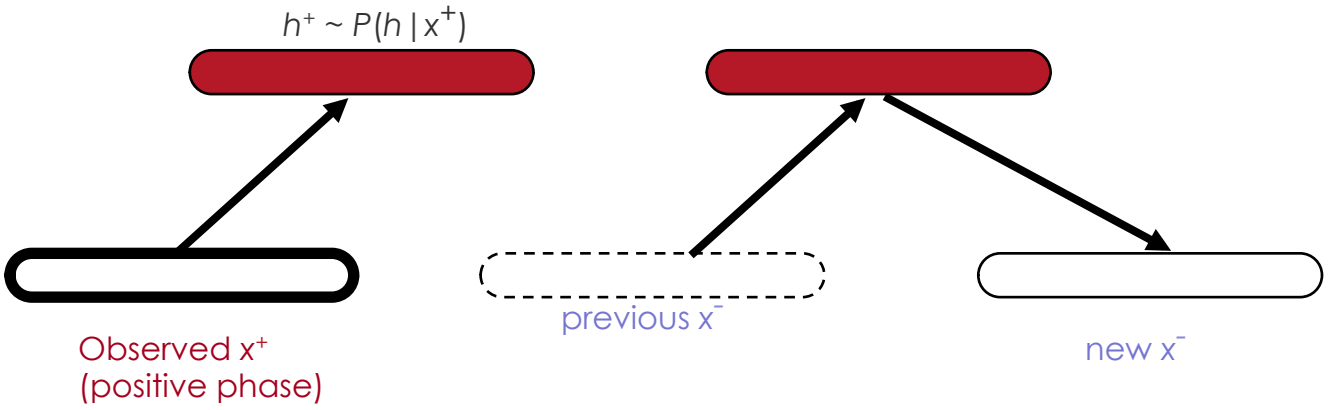


Run negative Gibbs chain in background while weights slo change (Younes 1999, Tieleman 2008):

- Guarantees (Younes 1999; Yuille 2005)
- If learning rate decreases in $1/t$,

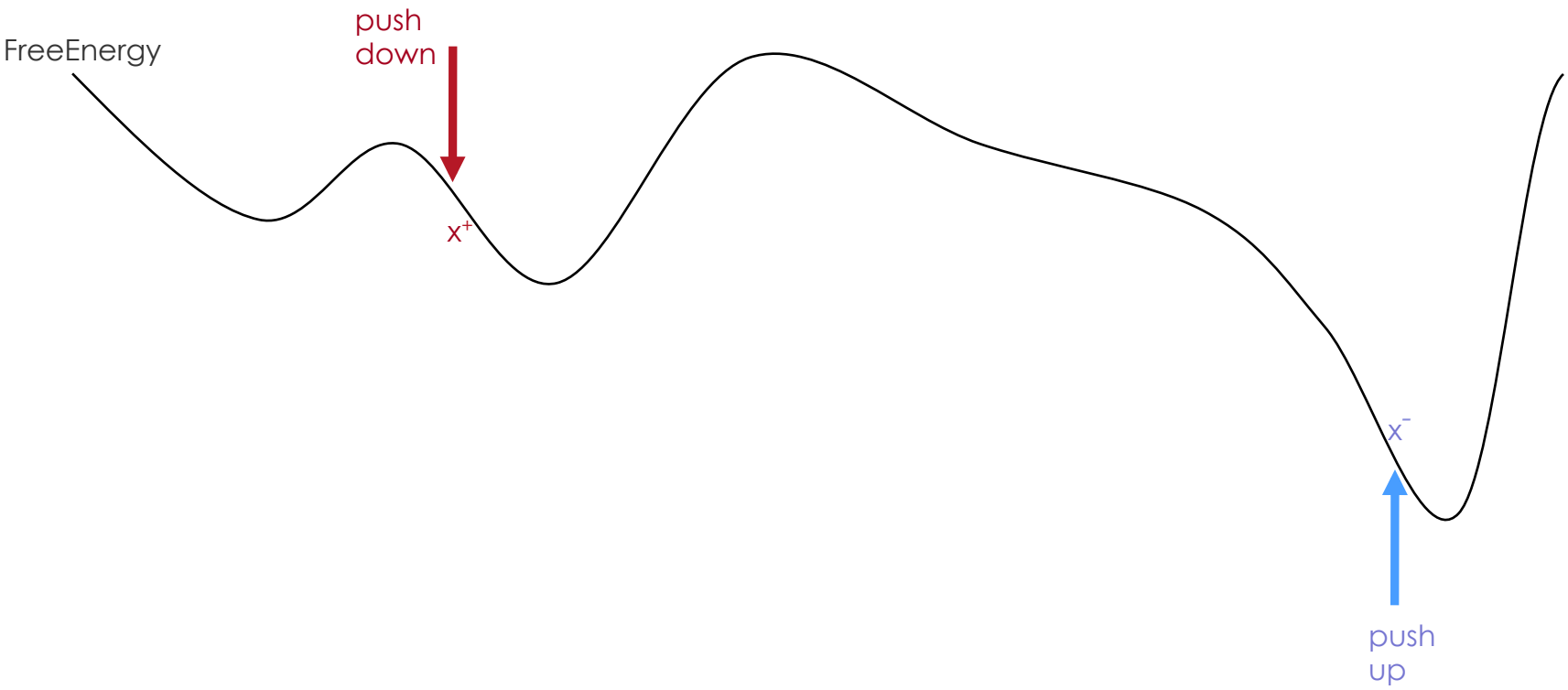
  chain mixes before parameters change too much,

  chain stays converged when parameters change



PCD/SML + large learning rate

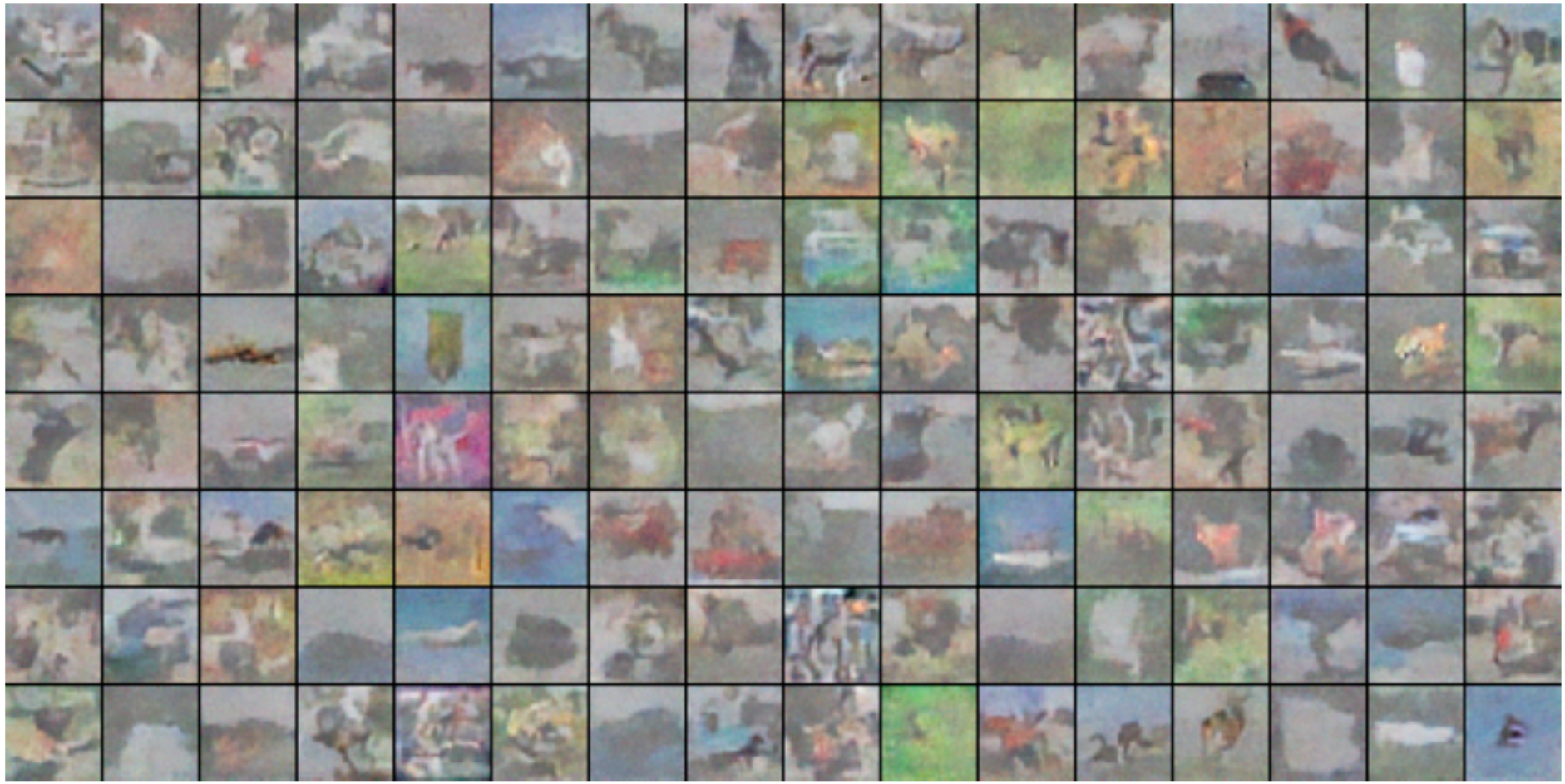
Negative phase samples quickly push up the energy of wherever they are and quickly move to another mode



Some RBM Variants

- Different energy functions and allowed values for the hidden and visible units:
  - Hinton et al 2006: binary-binary RBMs
  - Welling NIPS'2004: exponential family units
  - Ranzato & Hinton CVPR'2010: Gaussian RBM weaknesses (no conditional covariance), propose mcRBM
  - Ranzato et al NIPS'2010: mPoT, similar energy function
  - Courville et al ICML'2011: spike-and-slab RBM

Convolutionally Trained Spike & Slab RBMs Samples



ssRBM is not Cheating

Samples from μ-ssRBM:

Nearest examples in CIFAR:
(least square dist.)

Spike & Slab RBMs

$$E(v,s,h) = -\sum_{i=1}^{N} v^T W_i s_i h_i \; + \frac{1}{2} v^T \left(\Lambda + \sum_{i=1}^{N} \Phi_i h_i\right) v$$

$$+ \frac{1}{2}\sum_{i=1}^{N} \alpha_i s_i^2 \; - \sum_{i=1}^{N} \alpha_i \mu_i s_i h_i \; - \sum_{i=1}^{N} b_i h_i + \sum_{i=1}^{N} \alpha_i \mu_i^2 h_i,$$

Model conditional covariance of pixels (given hidden units)

$$C_{v|h} = \left(\Lambda + \sum_{i=1}^{N} \Phi_i h_i - \sum_{i=1}^{N} \alpha_i^{-1} h_i W_i W_i^T\right)^{-1}$$

Hidden representation decomposed into a product s*h, h is binary, s is real

s*h is often 0 (naturally sparse)

Spike & Slab RBMs

$$P(h_i = 1 \mid v) = \sigma(\hat{b}_i - \frac{1}{2}(v - \xi_{v|h_i})^T C_{v|h_i}^{-1}(v - \xi_{v|h_i}))$$

$$p(s \mid v, h) = \prod_{i=1}^{N} \mathcal{N}\left(\left(\alpha_i^{-1} v^T W_i + \mu_i\right) h_i \ , \ \alpha_i^{-1}\right)$$

$$p(v \mid s, h) = \mathcal{N}\left(C_{v|s,h} \sum_{i=1}^{N} W_i s_i h_i \ , \ C_{v|s,h}\right)$$

Can use efficient 3-way Gibbs sampling