Learning Deep Architectures

Yoshua Bengio, U. Montreal

UAI 2009

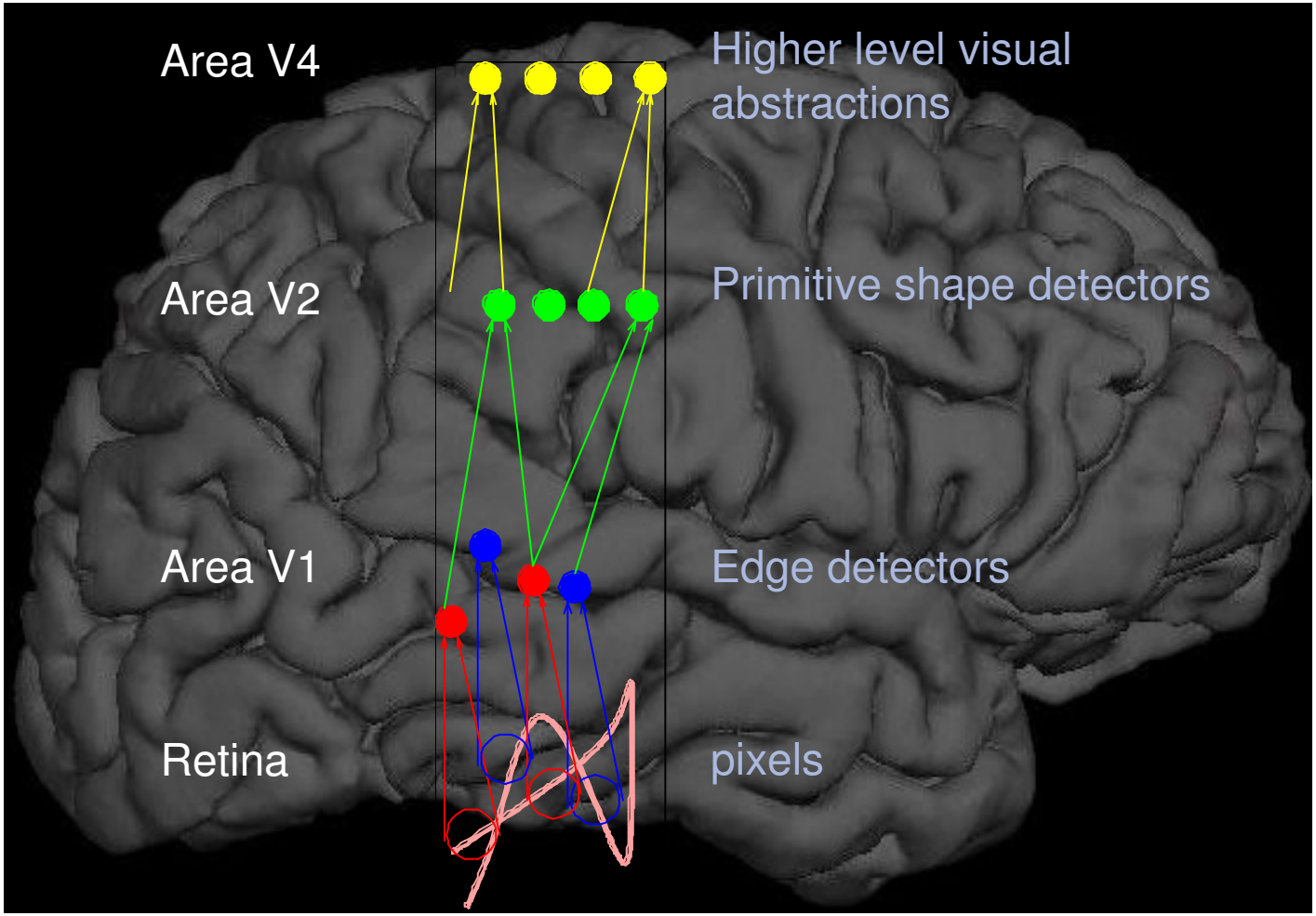
June 19th, 2009, Montreal

Thanks to: Aaron Courville, Pascal Vincent, Dumitru Erhan, Olivier Delalleau, Olivier Breuleux, Yann LeCun, Guillaume Desjardins, Pascal Lamblin, James Bergstra, Nicolas Le Roux, Max Welling, Myriam Côté, Jérôme Louradour, Ronan Collobert, Jason Weston

Deep Motivations

- Brains have a deep architecture
- Humans organize their ideas hierarchically, through composition of simpler ideas
- Unsufficiently deep architectures can be exponentially inefficient
- Distributed (possibly sparse) representations are necessary to achieve non-local generalization, exponentially more efficient than 1-of-N enumeration latent variable values
- Multiple levels of latent variables allow combinatorial sharing of statistical strength

Deep Architecture in the Brain



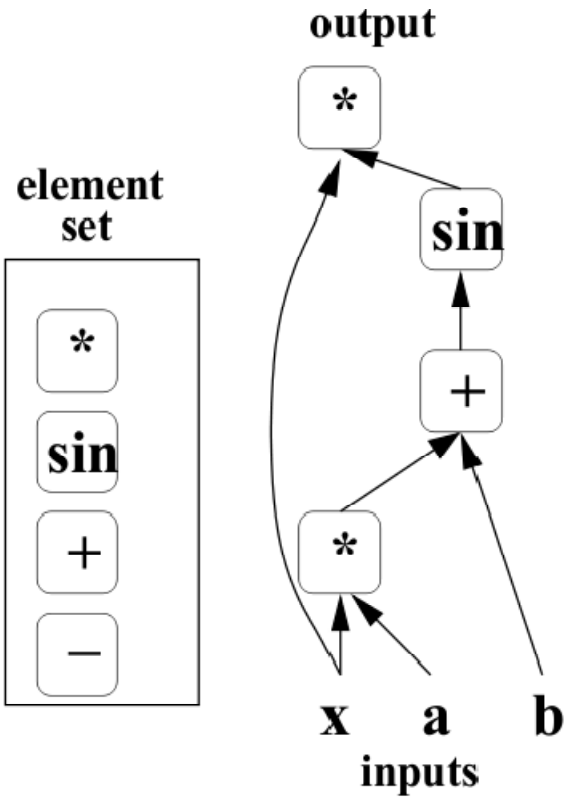
Deep Architecture in our Mind

- Humans organize their ideas and concepts hierarchically
- Humans first learn simpler concepts and then compose them to represent more abstract ones
- Engineers break-up solutions into multiple levels of abstraction and processing

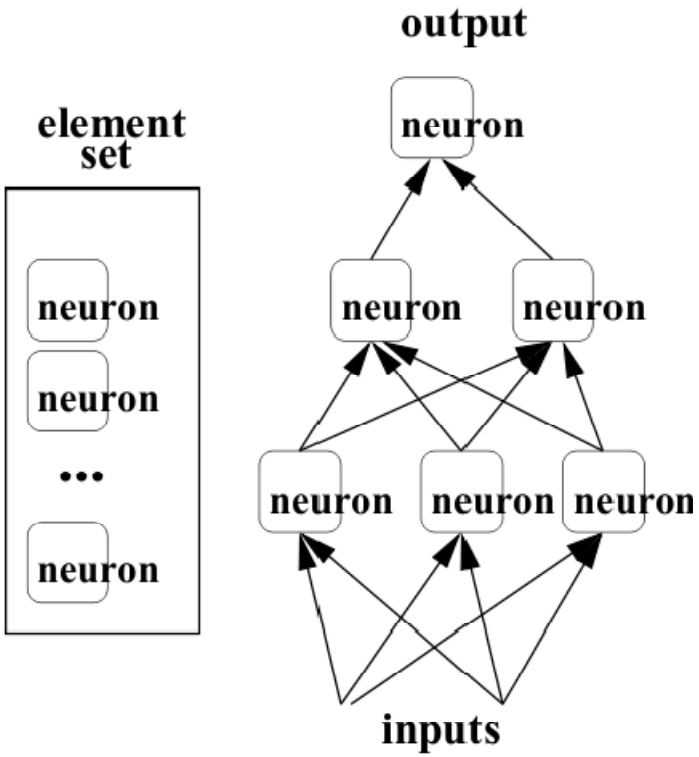
It would be nice to learn / discover these concepts

(knowledge engineering failed because of poor introspection?)

Architecture Depth



Depth = 4



Depth = 3

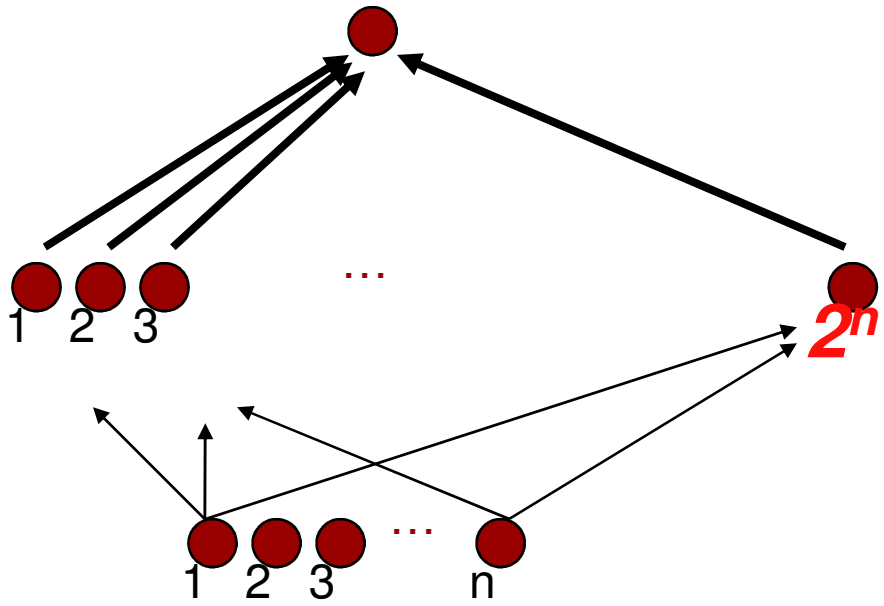
Good News, Bad News

Theoretical arguments: deep architectures can be

2 layers of { logic gates / formal neurons / RBF units } = universal approximator

Theorems for all 3:
(Hastad et al 86 & 91, Bengio et al 2007)

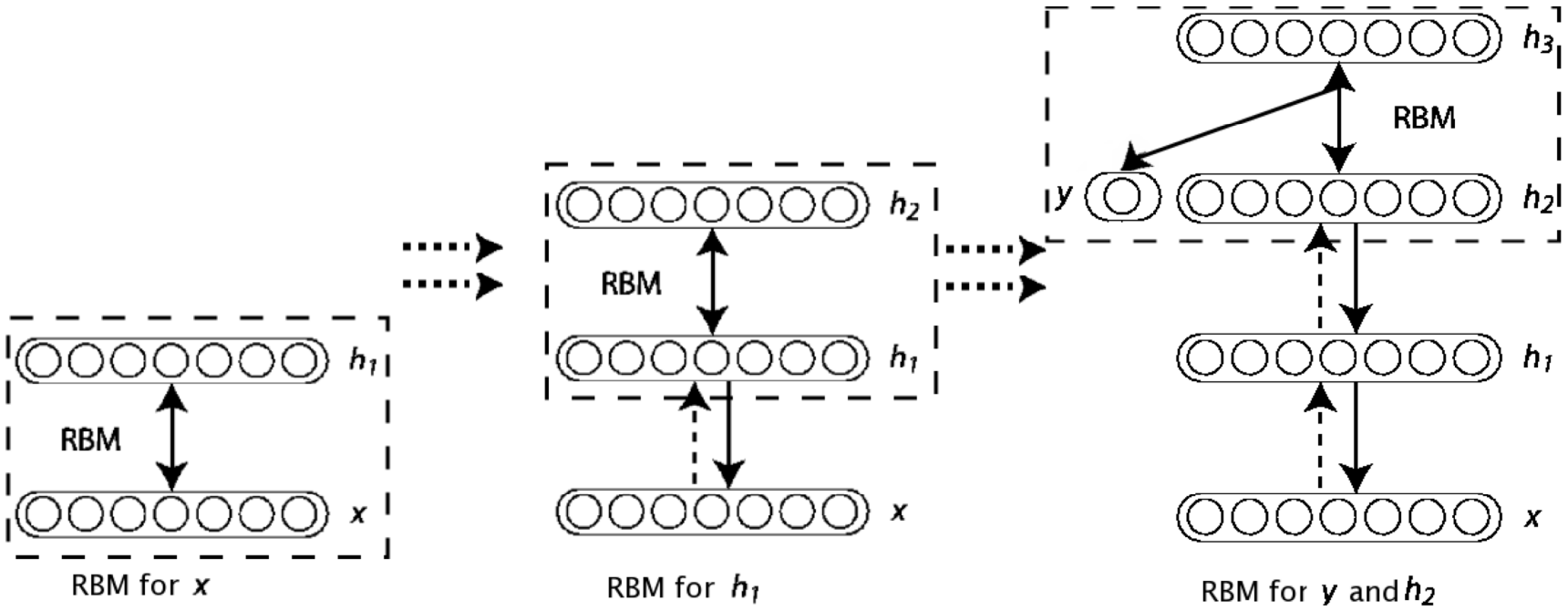
Functions representable compactly with k layers may require exponential size with k-1 layers



The Deep Breakthrough

- Before 2006, training deep architectures was unsuccessful, except for convolutional neural nets
- Hinton, Osindero & Teh « A Fast Learning Algorithm for Deep Belief Nets », *Neural Computation*, 2006
- Bengio, Lamblin, Popovici, Larochelle « Greedy Layer-Wise Training of Deep Networks », *NIPS'2006*
- Ranzato, Poultney, Chopra, LeCun « Efficient Learning of Sparse Representations with an Energy-Based Model », *NIPS'2006*

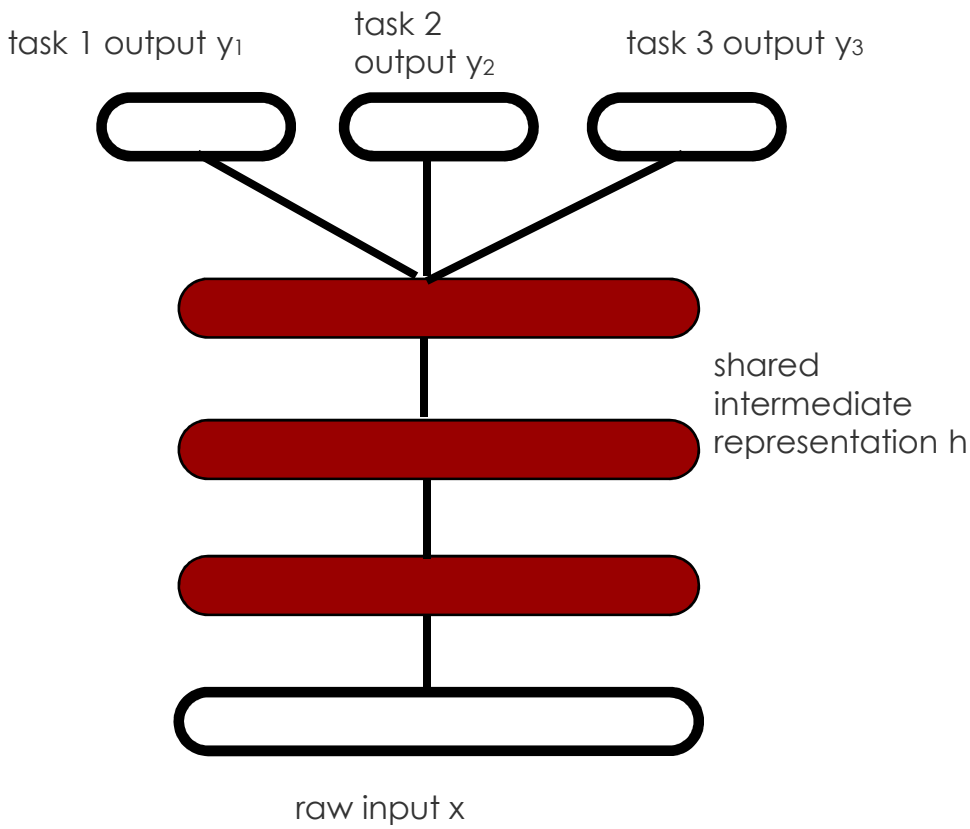
Greedy Layer-Wise Pre-Training



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

Deep Architectures and Sharing Statistical Strength, Multi-Task Learning

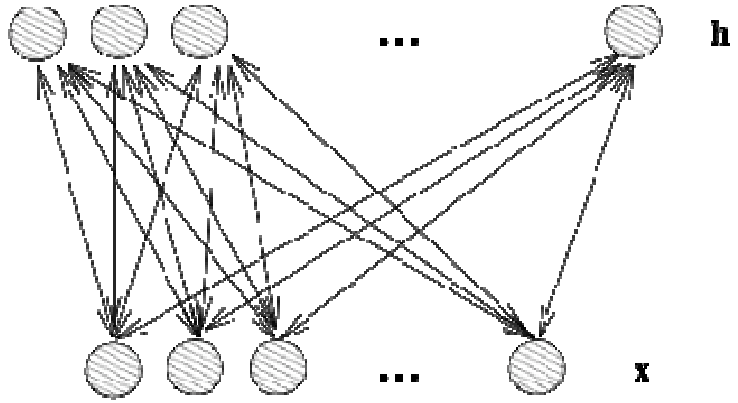
- Generalizing better to new tasks is crucial to approach AI
- Deep architectures learn good intermediate representations that can be shared across tasks
- A good representation is one that makes sense for many tasks



Restricted Boltzmann Machines

- The most popular building block for deep architectures
- Bipartite undirected graphical model.

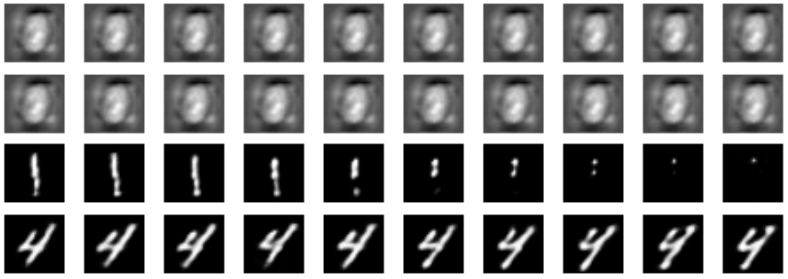
x=observed, h=hidden



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

- P(h | x) and P(x | h) factorize:
  - Easy inference
  - Convenient Gibbs sampling x→h→x→h…
- In practice, Gibbs sampling does not always mix well…

RBM trained by CD on MNIST



Chains from random state

Chains from real digit

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' and 'negative phase'

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

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

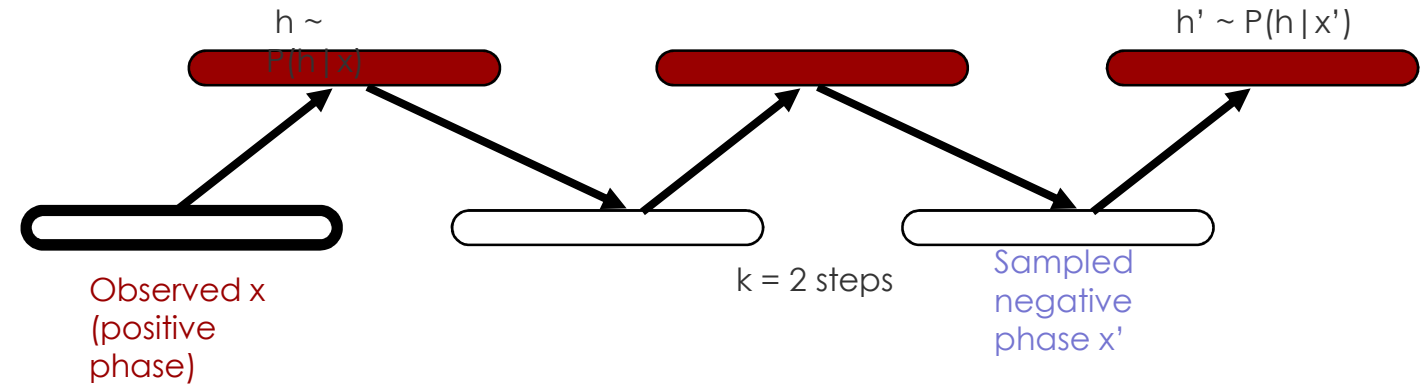
- In RBMs, easy to sample or sum over h | x:
- Difficult part: sampling from P(x), typically with a Markov chain

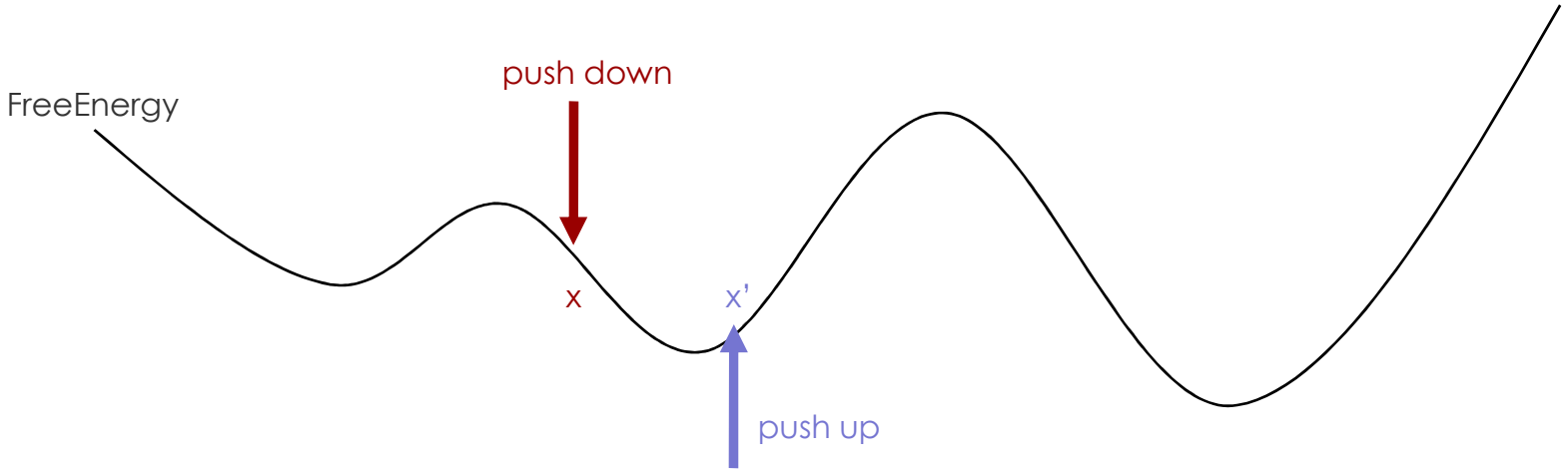
Training RBMs

- Contrastive Divergence (CD-k): start negative Gibbs chain at observed x, run k Gibbs steps.
- 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 (see Max Welling's ICML, UAI and workshop talks)
- Tempered MCMC: use higher temperature to escape modes

Contrastive Divergence

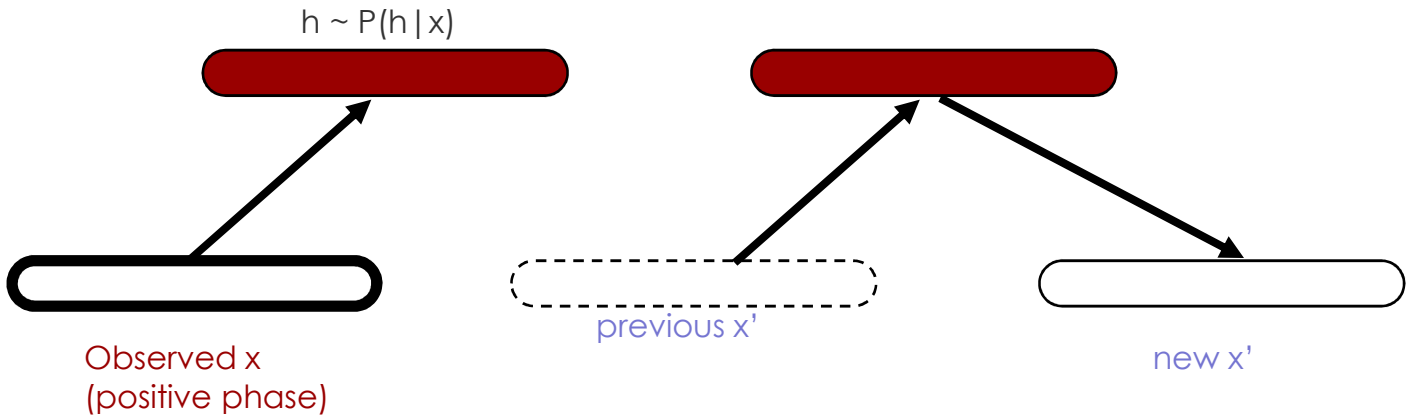
- Contrastive Divergence (CD-k): start negative phase block Gibbs chain at observed x, run k Gibbs steps (Hinton 2002)





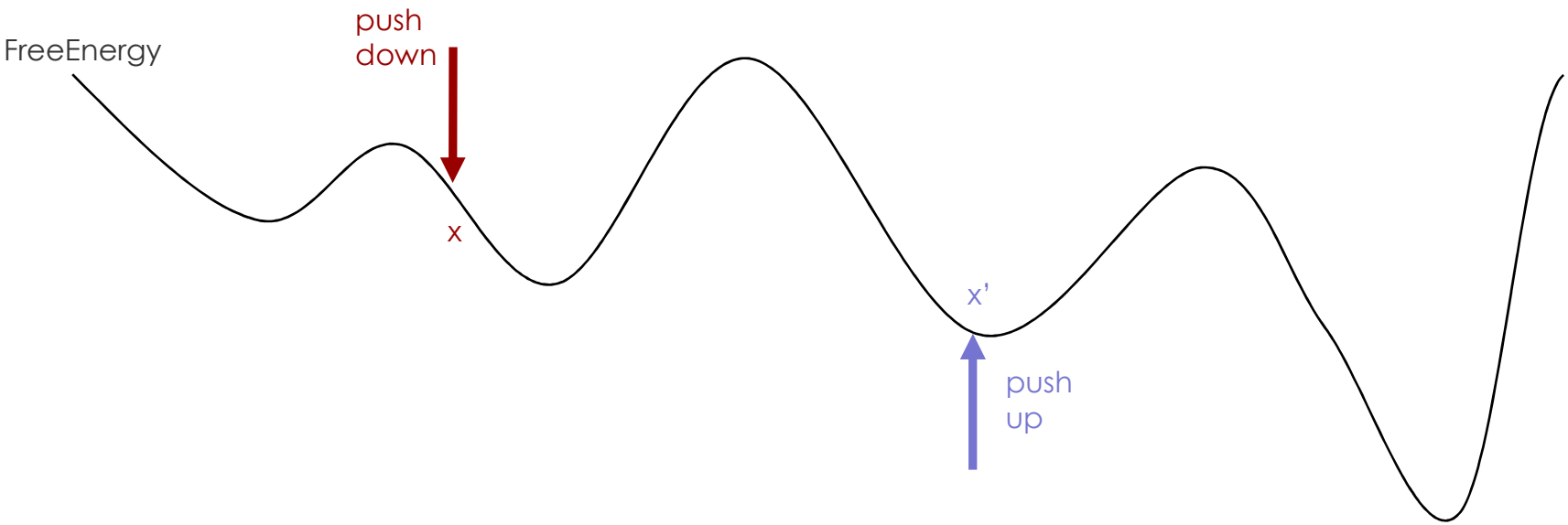
Persistent CD

- Persistent CD (PCD): run negative Gibbs chain in background while weights slowly change (Younes 2000, Tieleman 2008)
  - Guarantees (Younes 89,2000; Yuille 2004)
  - If learning rate decreases in 1/t, chain mixes before parameters change too much, chain stays converged when parameters change.



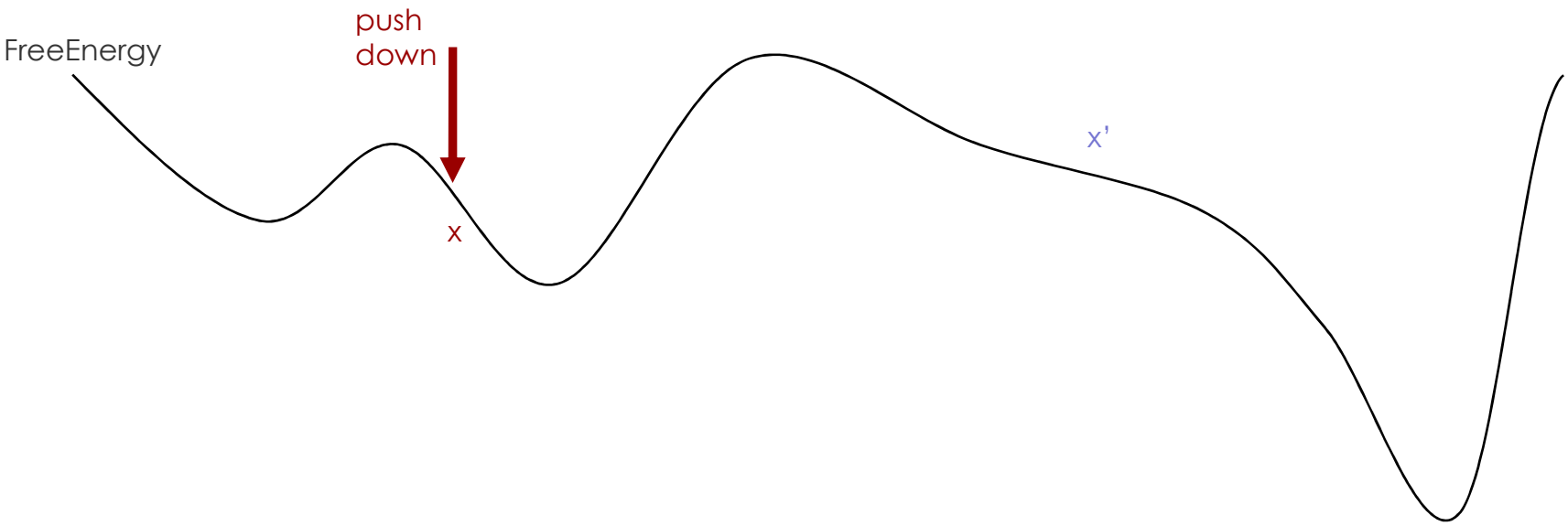
Persistent CD with large learning rate

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



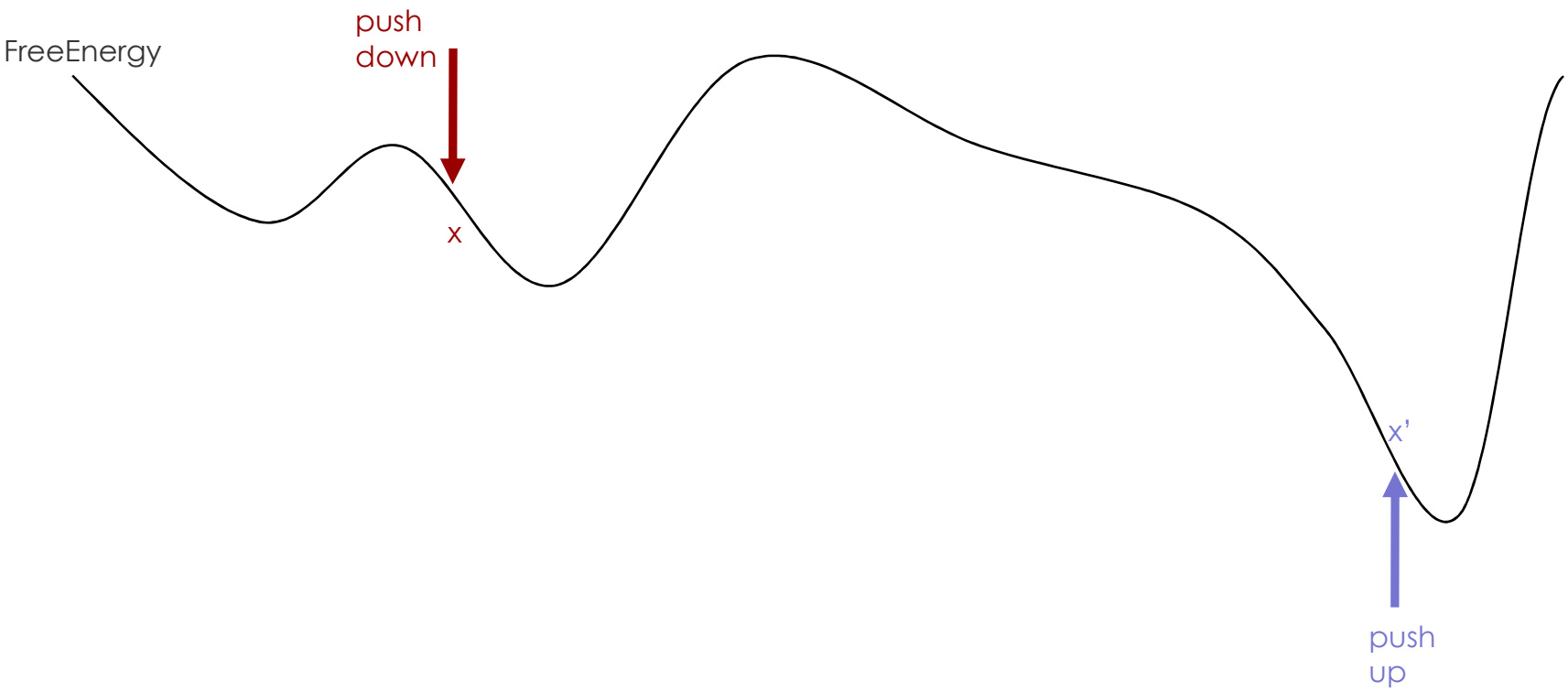
Persistent CD with large step size

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



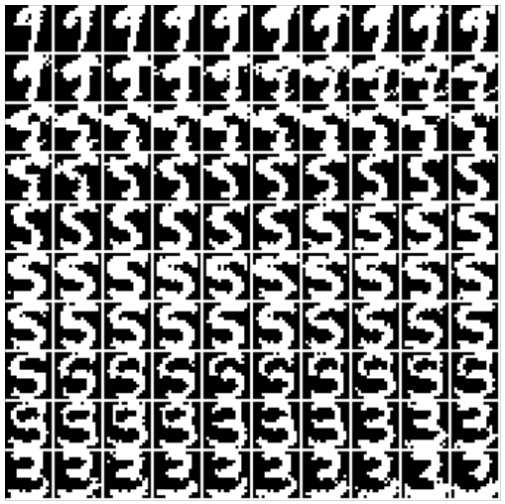
Persistent CD with large learning rate

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

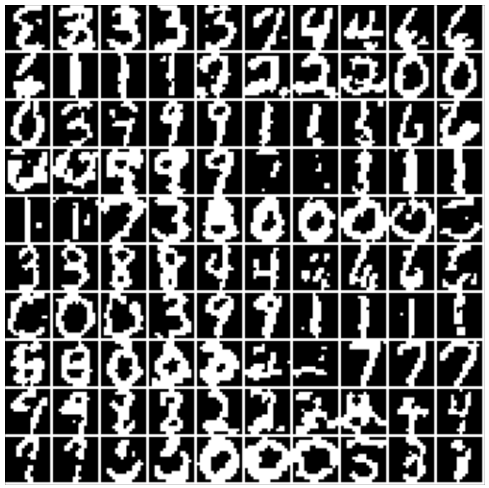


Fast Persistent CD and Herding

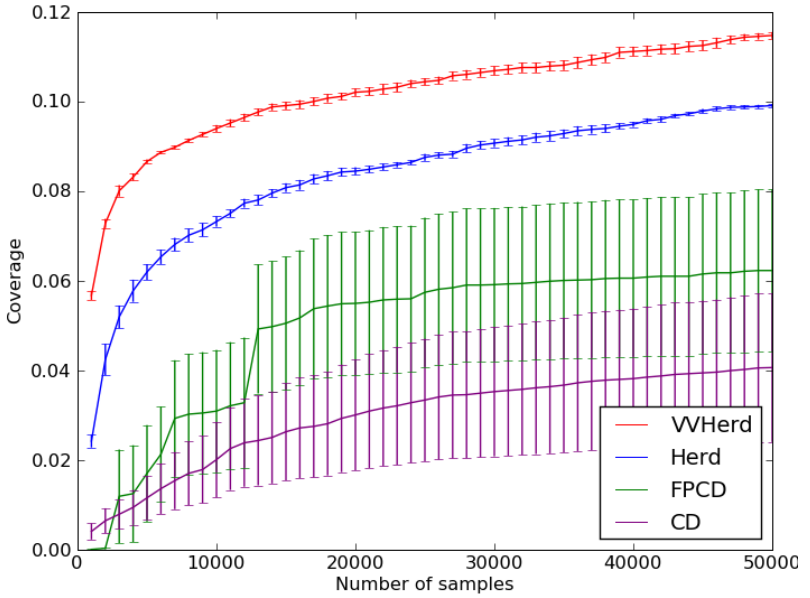
- Exploit **impressively faster mixing** achieved when parameters change quickly (large learning rate) while sampling
- Fast PCD: two sets of weights, one with a large learning rate only used for negative phase, quickly exploring modes
- Herding (see Max Welling's ICML, UAI and workshop talks): 0-temperature MRFs and RBMs, only use fast weights



FPCD



Herding



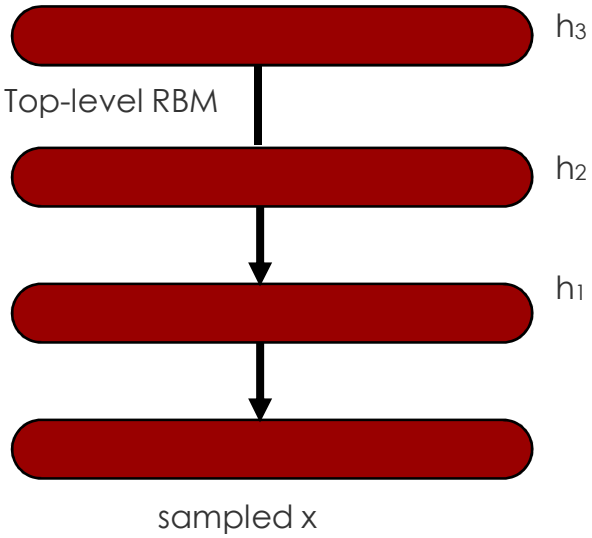
Tempered MCMC

- Annealing from high-temperature worked well for estimating log-likelihood (AIS)
- Consider multiple chains at different temperatures and reversible swaps between adjacent chains
- Higher temperature chains can escape modes
- Model samples are from T=1

	Sample generation procedure		
Training procedure	TMCMC	Gibbs (random start)	Gibbs (test start)
TMCMC	**215.45** $\pm$ 2.24	88.43 $\pm$ 2.75	60.04 $\pm$ 2.88
PCD	44.70 $\pm$ 2.51	-28.66 $\pm$ 3.28	-175.08 $\pm$ 2.99
CD	-2165 $\pm$ 0.53	-2154 $\pm$ 0.63	-842.76 $\pm$ 6.17

Deep Belief Networks

- Sampling:
  - Sample from top RBM
  - Sample from level k given k+1
- Easy approximate inference
- Training:
  - Variational bound justifies greedy layerwise training of RBMs
  - How to train all levels together?

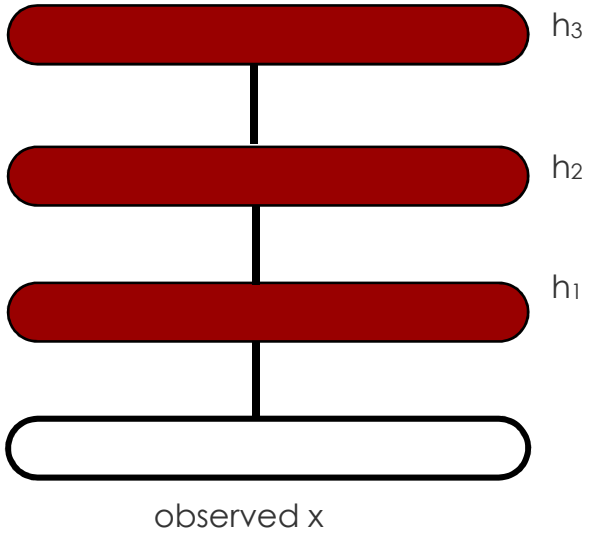


$$\log P(\mathbf{x}) \geq H_{Q(\mathbf{h}|\mathbf{x})} + \sum_{\mathbf{h}} Q(\mathbf{h}|\mathbf{x}) \left(\log P(\mathbf{h}) + \log P(\mathbf{x}|\mathbf{h})\right)$$

Deep Boltzmann Machines

(Salakhutdinov et al, AISTATS 2009, Lee et al, ICML 2009)

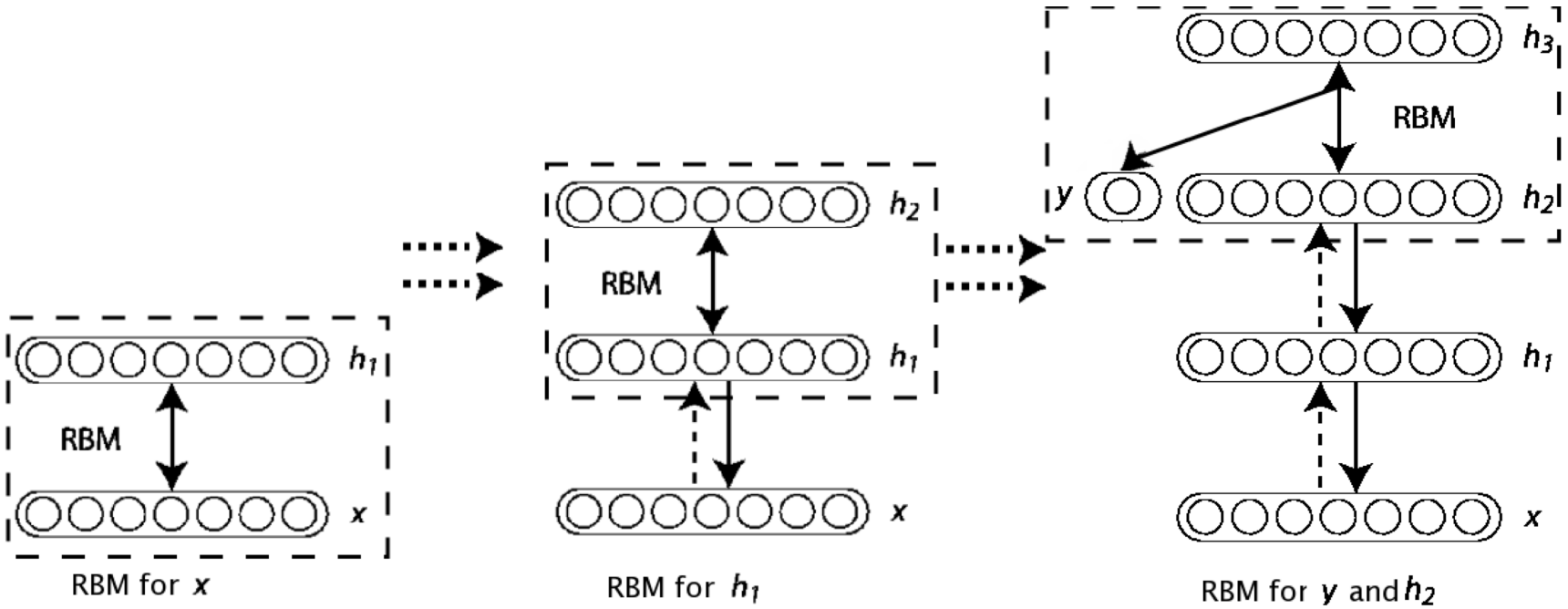
- Positive phase: variational approximation (mean-field)
- Negative phase: persistent chain
- Can (must) initialize from stacked RBMs
- Improved performance on MNIST from 1.2% to .95% error
- Can apply AIS with 2 hidden layers



Estimating Log-Likelihood

- RBMs: requires estimating partition function
  - Reconstruction error provides a cheap proxy
  - log Z tractable analytically for < 25 binary inputs or hidden
  - Lower-bounded (*how well?*) with Annealed Importance Sampling (AIS)
- Deep Belief Networks:
  - Extensions of AIS (Salakhutdinov & Murray, ICML 2008, NIPS 2008)
- Open question: efficient ways to monitor progress

Back to Greedy Layer-Wise Pre-Training

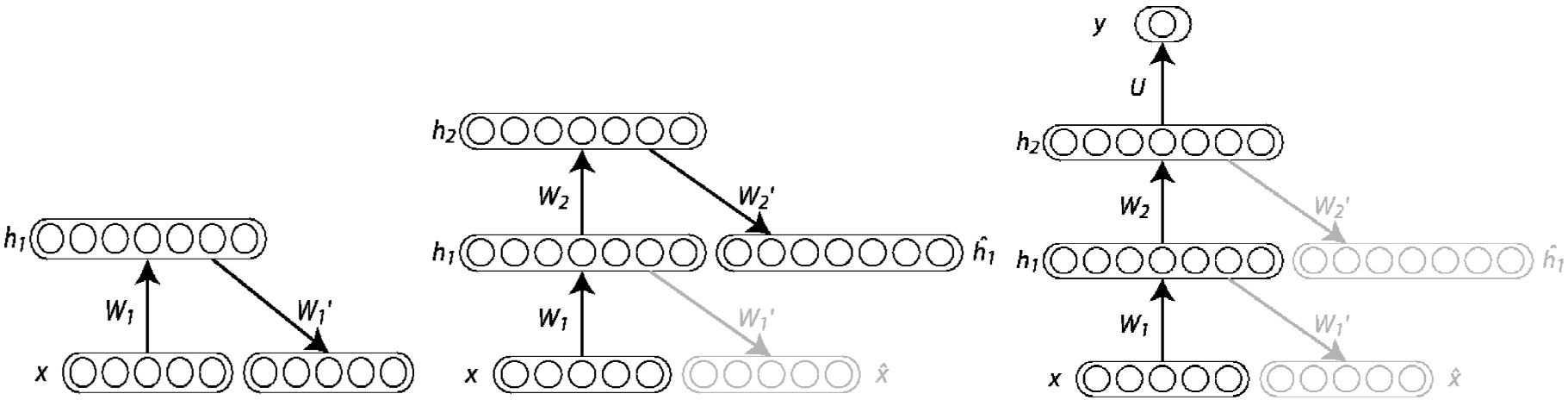


Stacking Restricted Boltzmann Machines (RBM) → Deep Belief Network (DBN)

Why are classifiers obtained from DBNs working so well?

- General principles?
- Would these principles work for other single-level algorithms?
- Why does it work?

Stacking Auto-Encoders



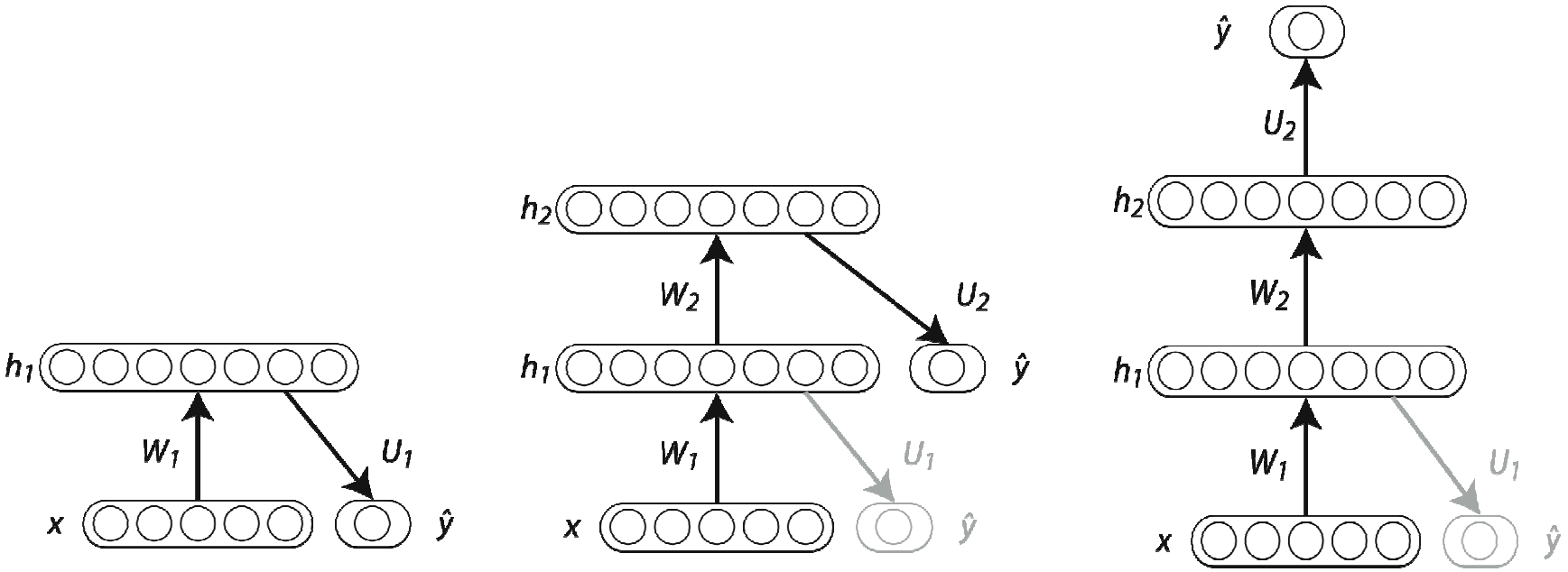
Auto-encoders and CD

RBM log-likelihood gradient can be written as converging expansion: CD-k = 2 k terms, reconstruction error ~ 1 term.

$$\frac{\partial \log P(x_1)}{\partial \theta} = \sum_{s=1}^{t-1} \left(E\left[\frac{\partial \log P(x_s|h_s)}{\partial \theta} \middle| x_1 \right] + E\left[\frac{\partial \log P(h_s|x_{s+1})}{\partial \theta} \middle| x_1 \right] \right) + E\left[\frac{\partial \log P(x_t)}{\partial \theta} \middle| x_1 \right]$$

(Bengio & Delalleau 2009)

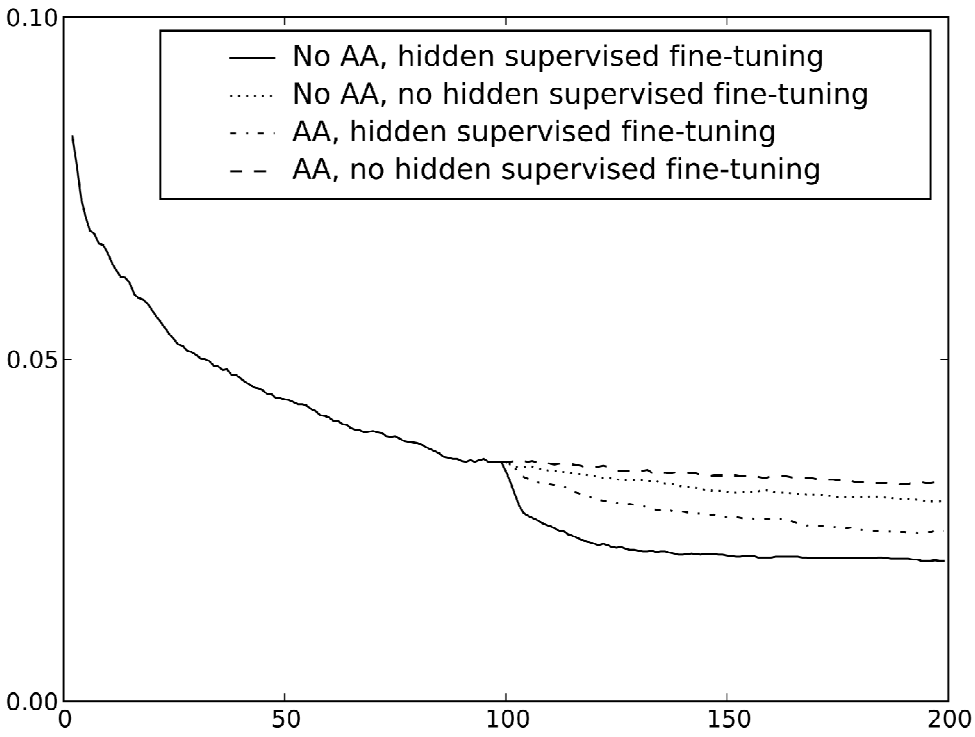
Greedy Layerwise Supervised Training



Generally worse than unsupervised pre-training but better than ordinary training of a deep neural network (Bengio et al. 2007).

Supervised Fine-Tuning is Important

- Greedy layer-wise unsupervised pre-training phase with RBMs or auto-encoders on MNIST
- Supervised phase with or without unsupervised updates, with or without fine-tuning of hidden layers
- Can train all RBMs at the same time, same results



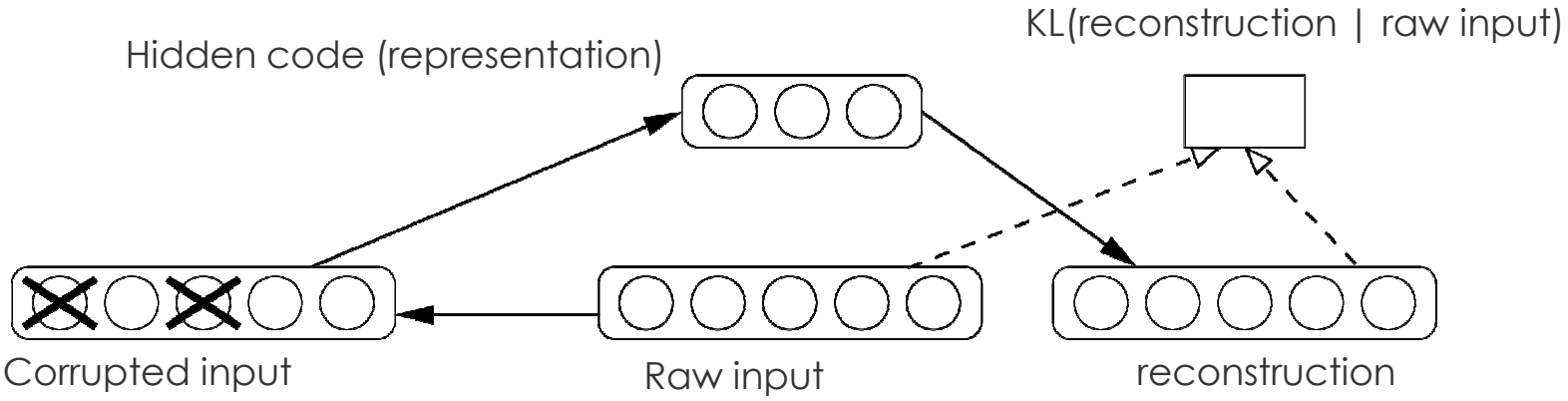
Sparse Auto-Encoders

- (Ranzato et al, 2007; Ranzato et al 2008)
- Sparsity penalty on the intermediate codes
- Like sparse coding but with efficient run-time encoder
- Sparsity penalty pushes up the free energy of all configurations (proxy for minimizing the partition function)
- Impressive results in object classification (convolutional nets):
  - MNIST: .5% error = record-breaking
  - Caltech-101: 65% correct = state-of-the-art (Jarrett et al, ICCV 2009)

similar results obtained with a convolutional DBN: (Lee et al, ICML'2009)

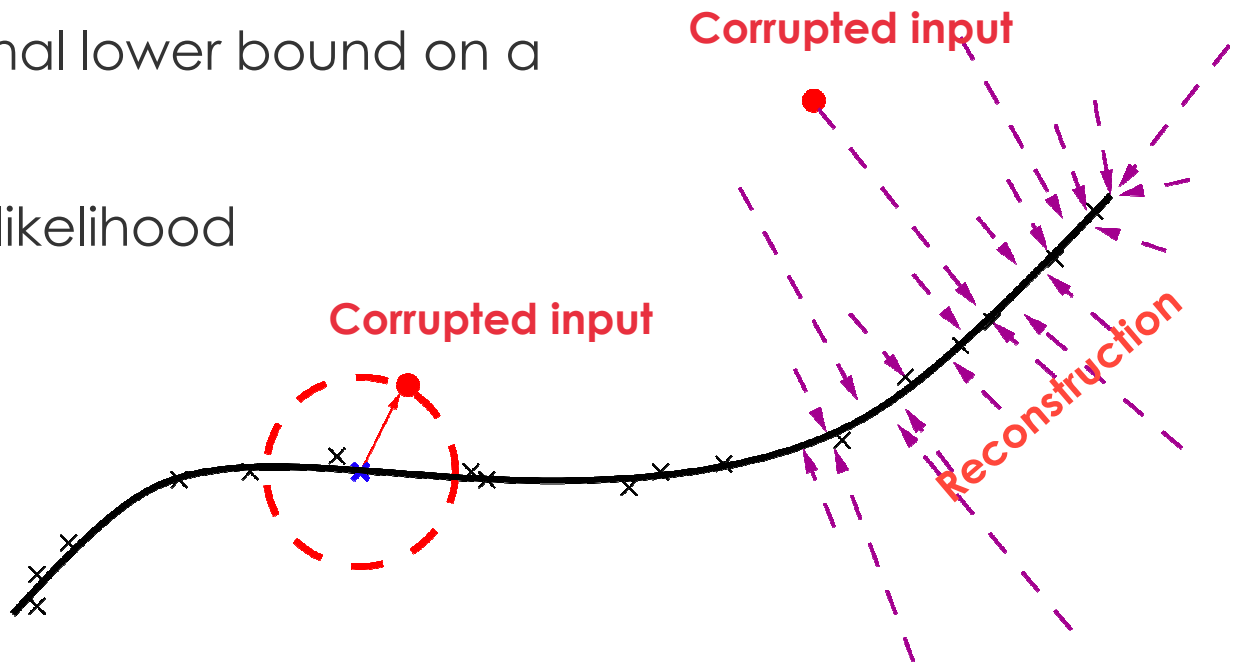
Denoising Auto-Encoder

- (Vincent et al, 2008)
- Corrupt the input
- Reconstruct the uncorrupted input



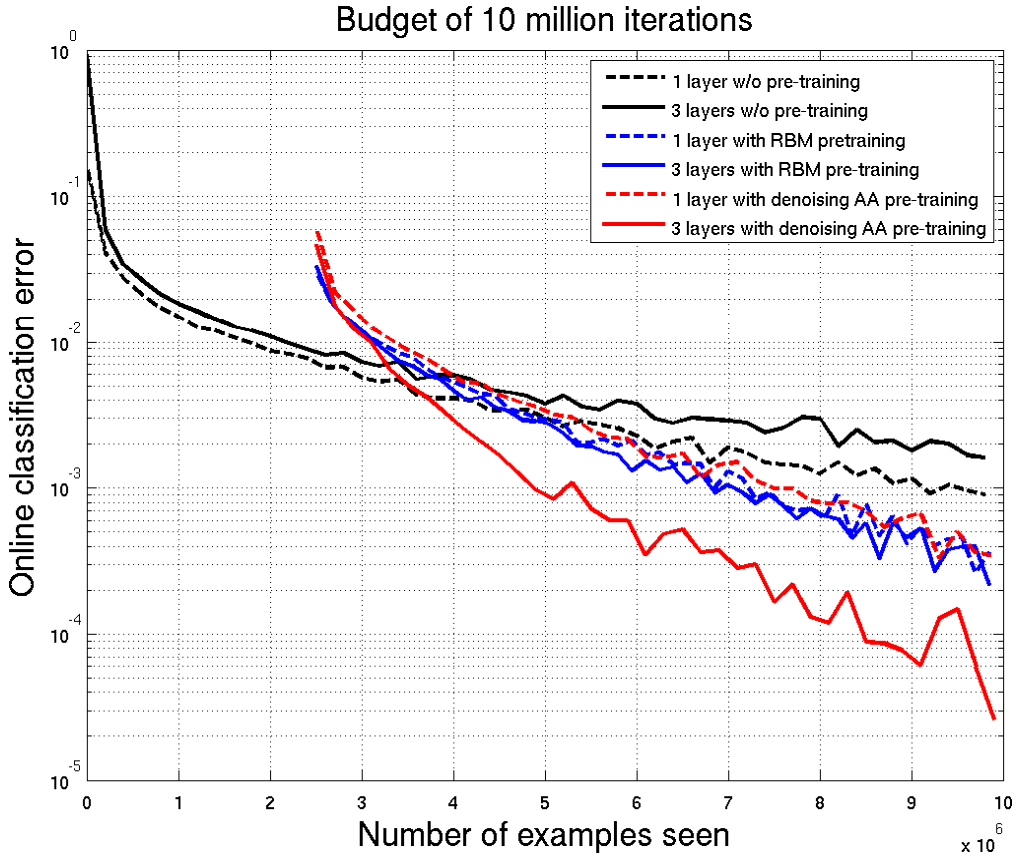
Denoising Auto-Encoder

- Learns a vector field towards higher probability regions
- Minimizes variational lower bound on a generative model
- Similar to pseudo-likelihood



Stacked Denoising Auto-Encoders

- No partition function, can measure training criterion
- Encoder & decoder: any parametrization
- Performs as well or better than stacking RBMs for usupervised pre-training



Infinite MNIST

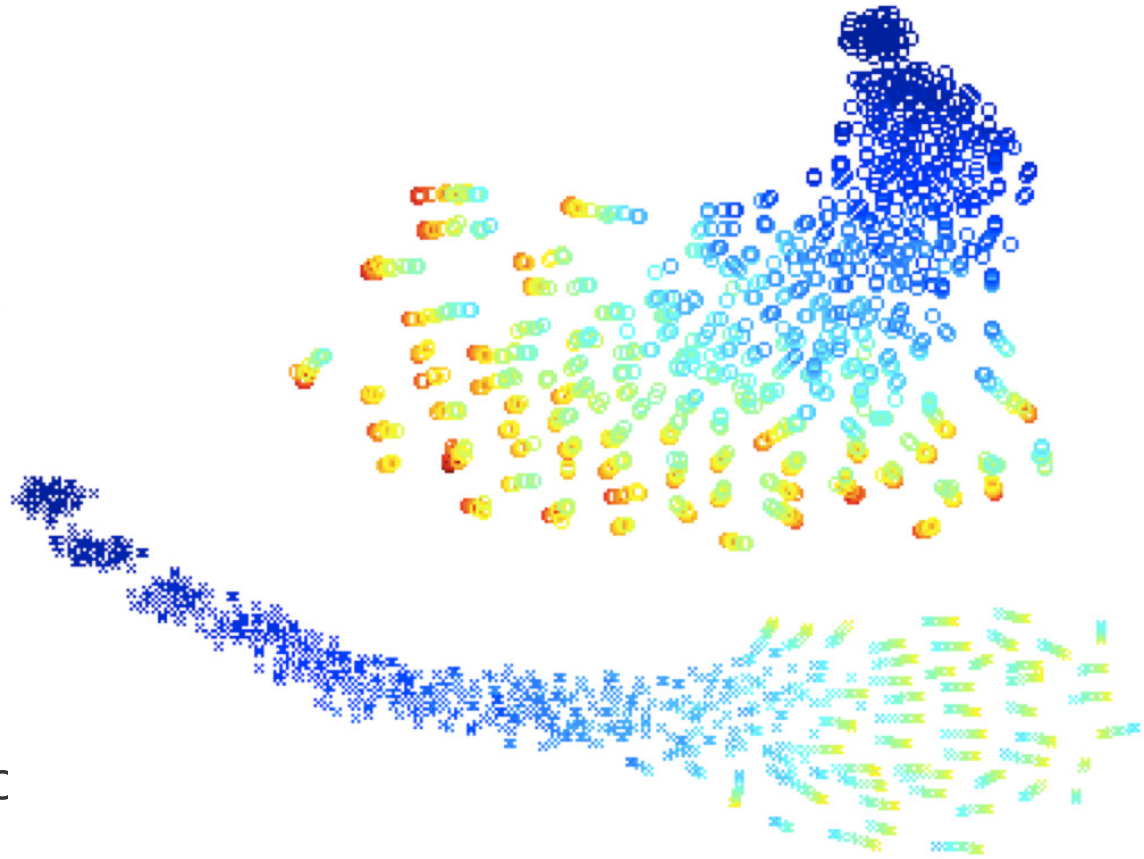
Why is Unsupervised Pre-Training Working So Well?

- Regularization hypothesis:
  - Unsupervised component forces model close to P(x)
  - Representations good for P(x) are good for P(y | x)

- Optimization hypothesis:
  - Unsupervised initialization near better local minimum of P(y | x)
  - Can reach lower local minimum otherwise not achievable by random initialization

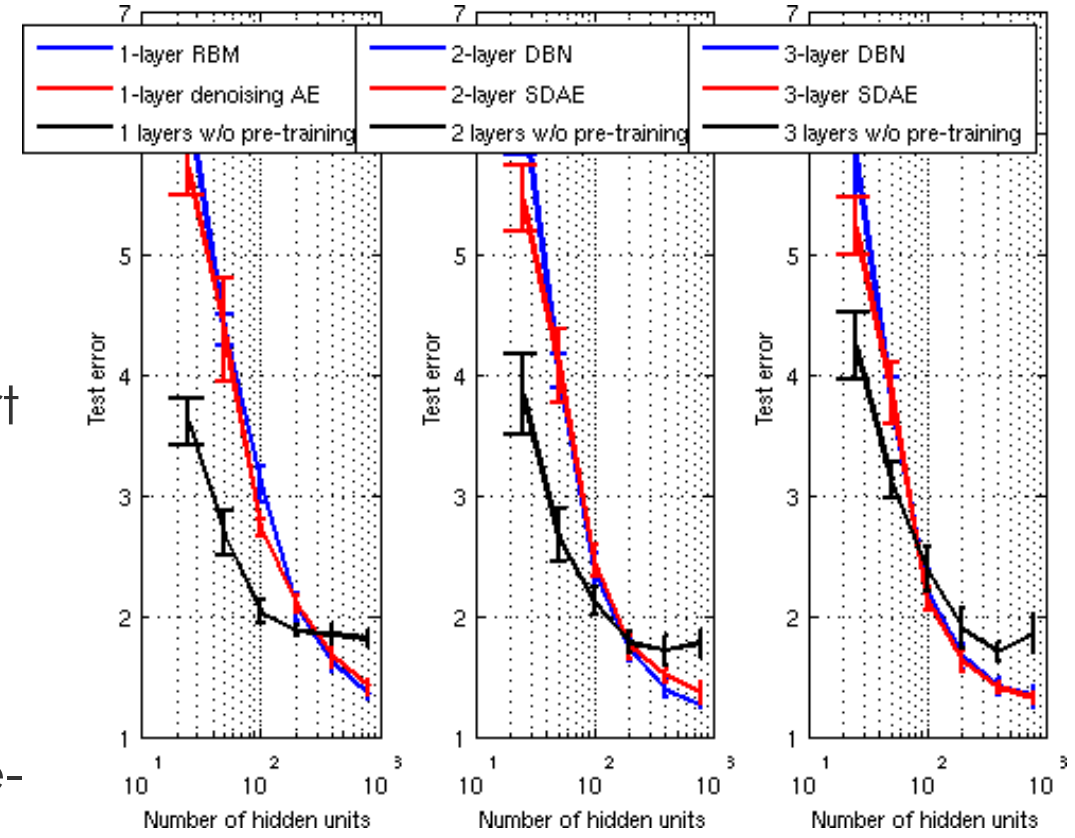
Learning Trajectories in Function Space

- Each point a model in function space
- Color = epoch
- Top: trajectories w/o pre-training
- Each trajectory converges in different local min.
- No overlap of regions with and w/o pre-training



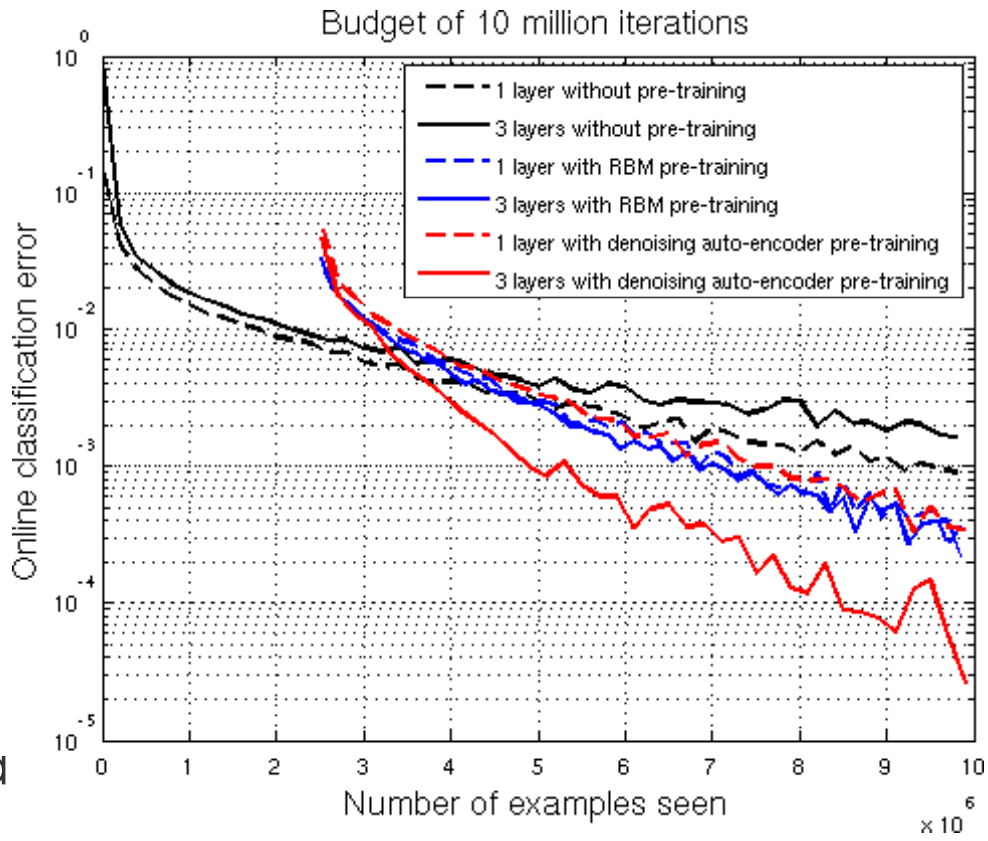
Unsupervised learning as regularizer

- Adding extra regularization (reducing # hidden units) hurts more the pre-trained models
- Pre-trained models have less variance wrt training sample
- Regularizer = infinite penalty outside of region compatible with unsupervised pre-training

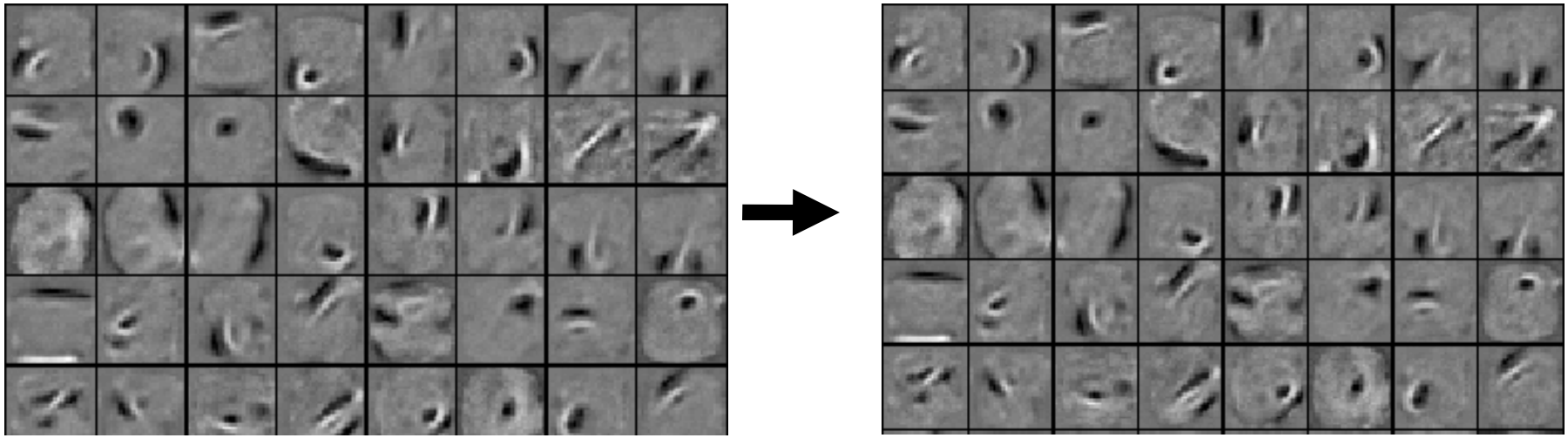


Better optimization of online error

- Both training and online error are smaller with unsupervised pre-training
- As # samples → ∞ training err. = online err. = generalization err.
- Without unsup. pre-training: can't exploit capacity to capture complexity in target function from training data



Learning Dynamics of Deep Nets

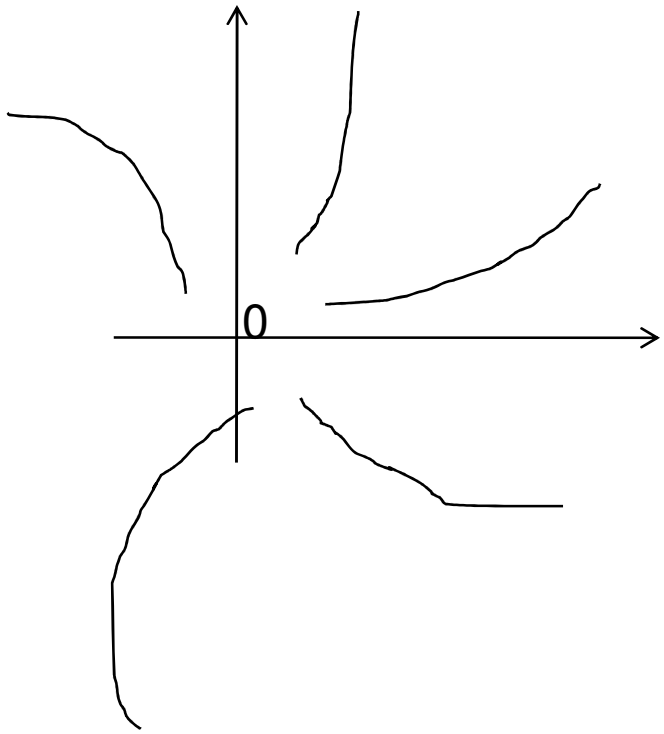


Before fine-tuning

After fine-tuning

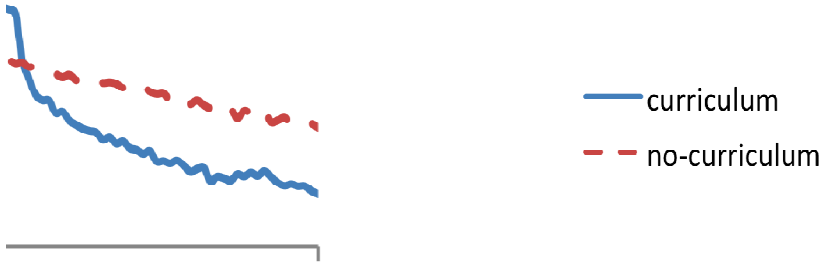
Learning Dynamics of Deep Nets

- As weights become larger, get trapped in basin of attraction ("quadrant" does not change)
- Initial updates have a crucial influence ("critical period"), explain more of the variance
- Unsupervised pre-training initializes in basin of attraction with good generalization properties



The order and selection of examples makes a difference

- Curriculum learning (Bengio et al, ICML'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?
- **Influencing learning dynamics can make a big difference**



Level-local learning is important

- Initializing each layer of an unsupervised deep Boltzmann machine helps a lot
- Initializing each layer of a supervised neural network as an RBM helps a lot
- Helps most the layers further away from the target
- Not just an effect of unsupervised prior
- Jointly training all the levels of a deep architecture is difficult
- Initializing using a level-local learning algorithm (RBM, auto-encoders, etc.) is a useful trick

Take-Home Messages

- Multiple levels of latent variables: potentially exponential gain in statistical sharing
- RBMs allow fast inference, stacked RBMs / auto-encoders have fast approximate inference
- Gibbs sampling in RBMs does not mix well, but sampling and learning can interact in surprisingly useful ways
- Unsupervised pre-training of classifiers acts like a strange regularizer with improved optimization of online error
- At least as important as the model: the inference approximations and the learning dynamics

Some Open Problems

- Why is it difficult to train deep architectures?
- What is important in the learning dynamics?
- How to improve joint training of all layers?
- How to sample better from RBMs and deep generative models?
- Monitoring unsupervised learning quality in deep nets?
- Other ways to guide training of intermediate representations?

THANK YOU FOR YOUR ATTENTION!

- Questions?
- Comments?