

Learning Deep Architectures

Yoshua Bengio, [U. Montreal](#)

CIFAR NCAP Summer School 2009

August 6th, 2009, Montreal

Main reference: “Learning Deep Architectures for AI”, Y. Bengio,
to appear in *Foundations and Trends in Machine Learning*, available on my web page.

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, Pierre-Antoine Manzagol, Ronan Collobert, Jason Weston

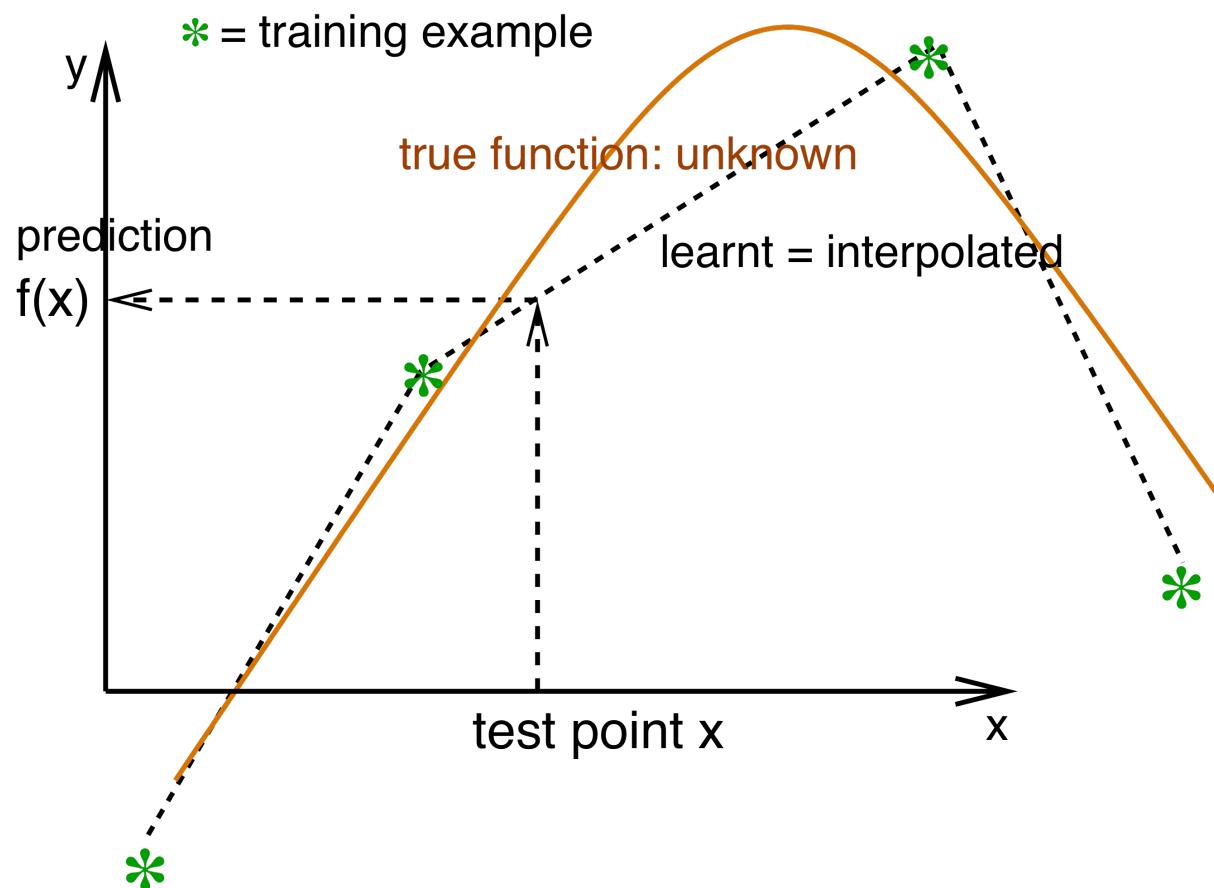
Deep Architectures Work Well

- Beating shallow neural networks on vision and NLP tasks
- Beating SVMs on visions tasks from pixels (and handling dataset sizes that SVMs cannot handle in NLP)
- Reaching state-of-the-art performance in NLP
- Beating deep neural nets without unsupervised component
- Learn visual features similar to V1 and V2 neurons

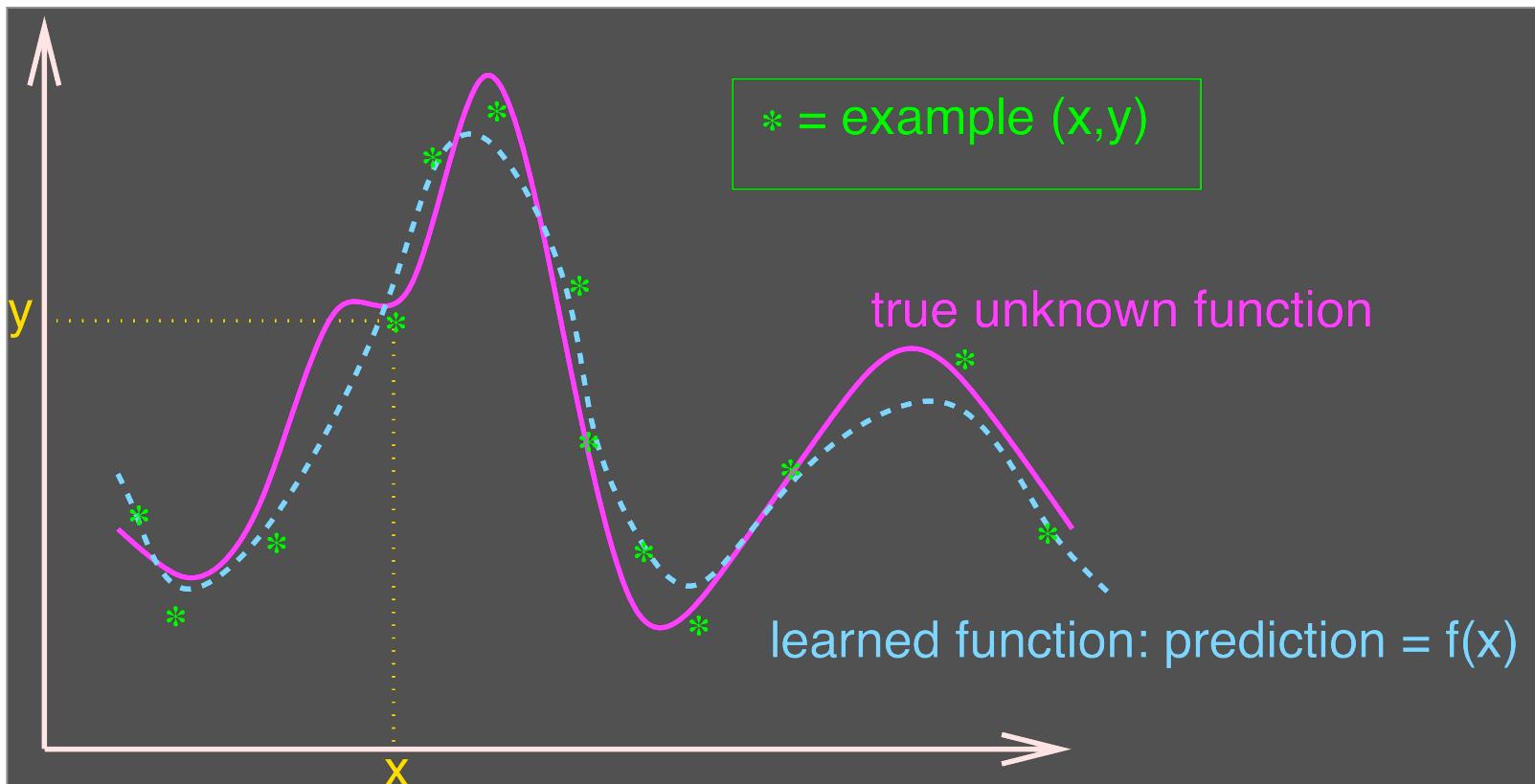
Deep Motivations

- Brains have a deep architecture
- Humans organize their ideas hierarchically, through composition of simpler ideas
- Insufficiently 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

Locally Capture the Variations

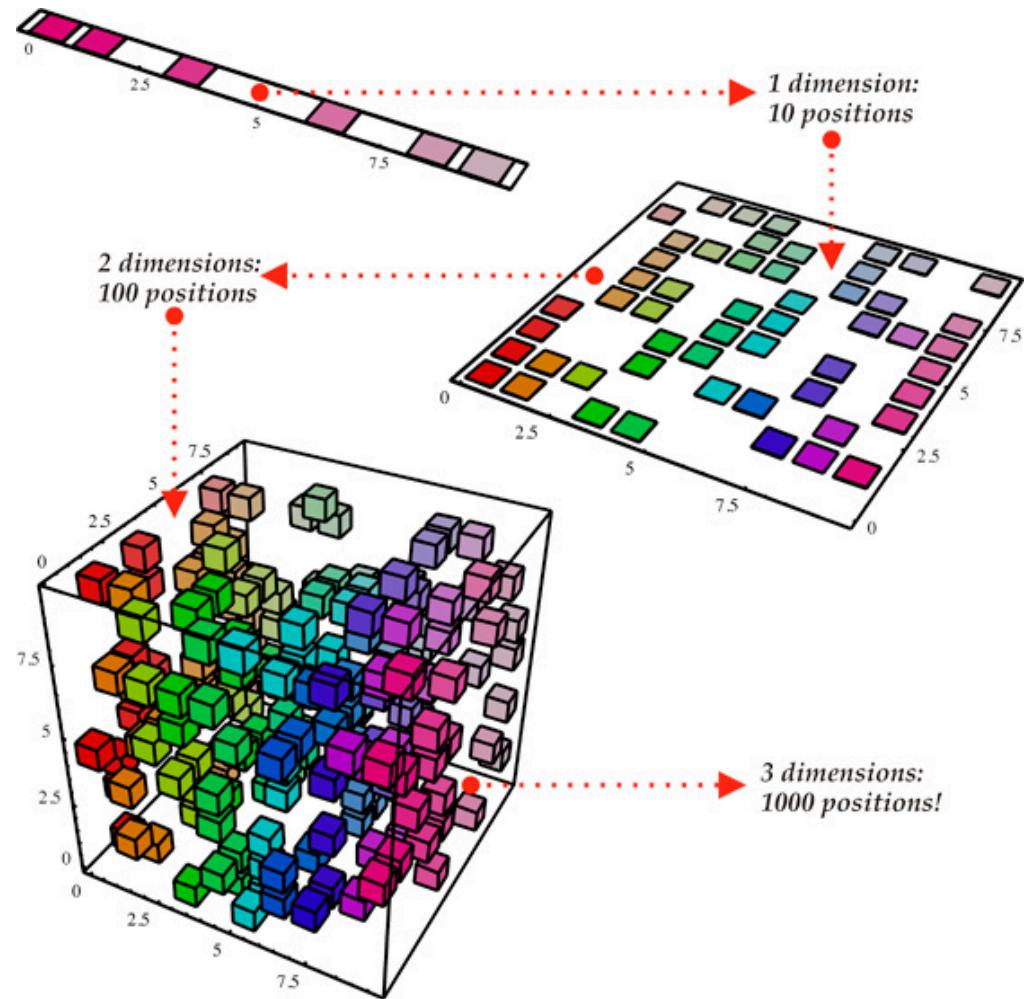


Easy with Few Variations



The Curse of Dimensionality

To generalise locally,
need representative
exemplars for all
possible variations!

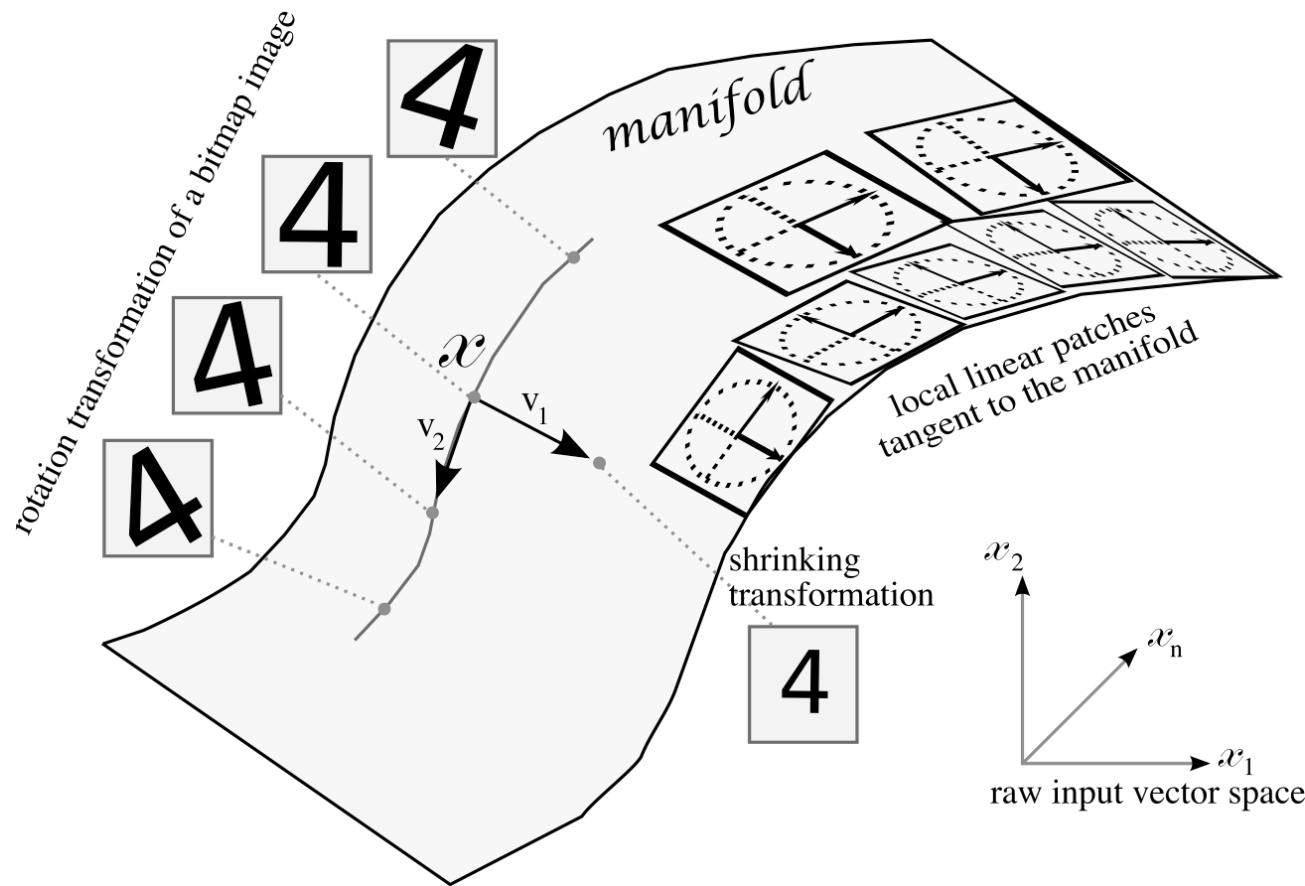


Limits of Local Generalization: Theoretical Results

(Bengio & Delalleau 2007)

- **Theorem:** Gaussian kernel machines need at least k examples to learn a function that has $2k$ zero-crossings along some line
- **Theorem:** For a Gaussian kernel machine to learn some maximally varying functions over d inputs require $O(2^d)$ examples

Curse of Dimensionality When Generalizing Locally on a Manifold



How to Beat the Curse of Many Factors of Variation?

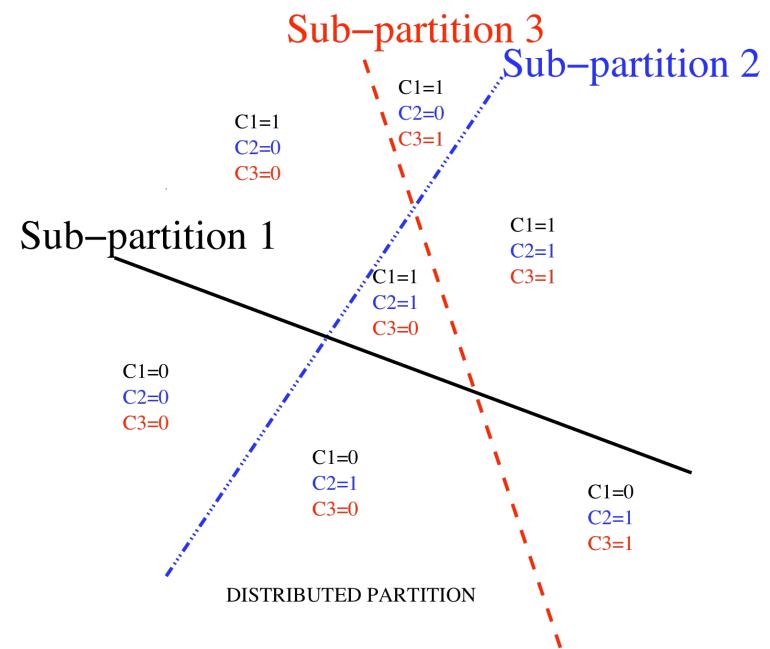
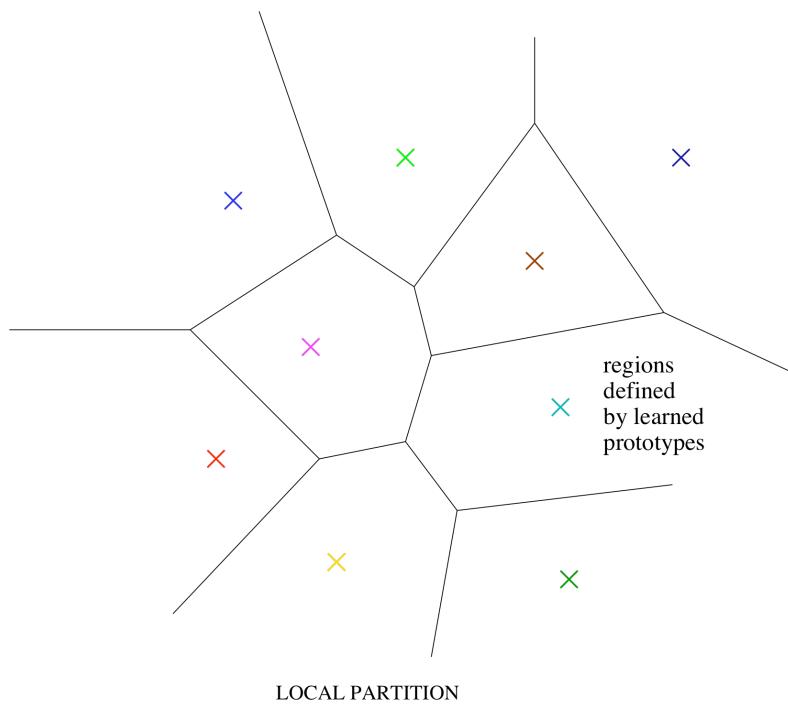
Compositionality: exponential gain in representational power

- Distributed representations
- Deep architecture

Distributed Representations

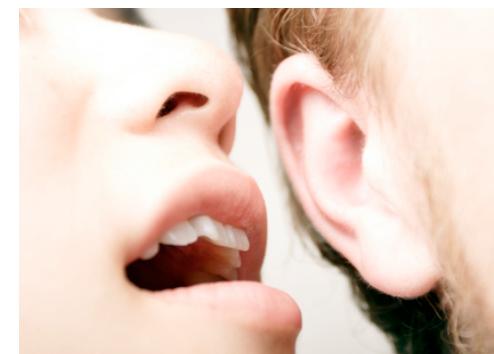
- Many neurons active simultaneously
- Input represented by the activation of a set of features that are not mutually exclusive
- Can be **exponentially more efficient** than local representations

Local vs Distributed

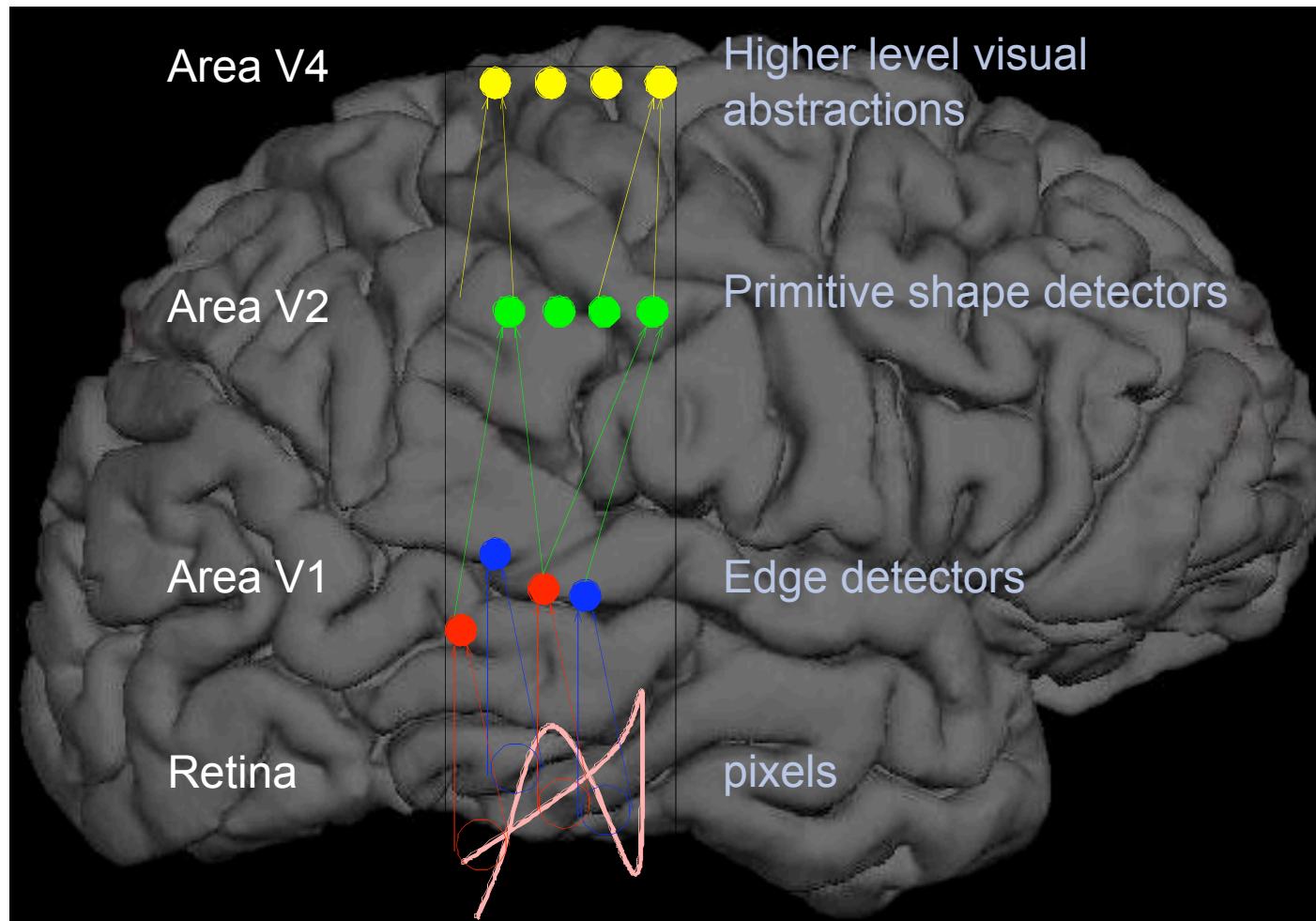


Neuro-cognitive inspiration

- Brains use a distributed representation
- Brains use a deep architecture
- Brains heavily use unsupervised learning
- Brains learn simpler tasks first
- Human brains developed with society / culture / education

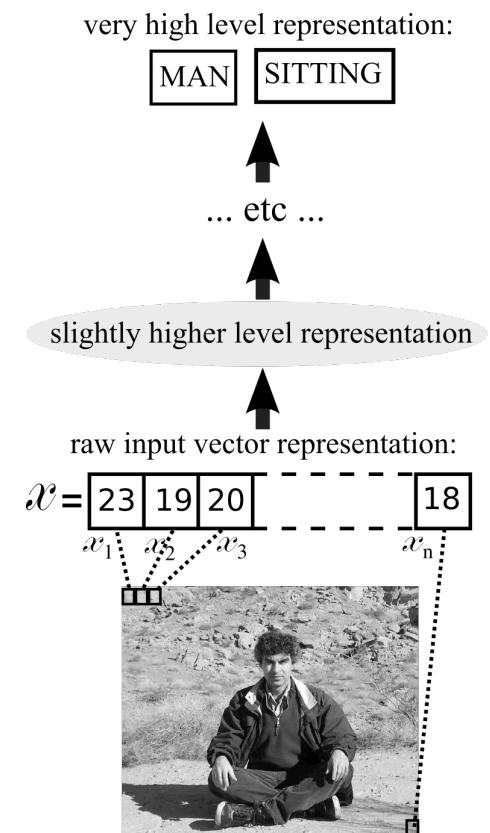


Deep Architecture in the Brain



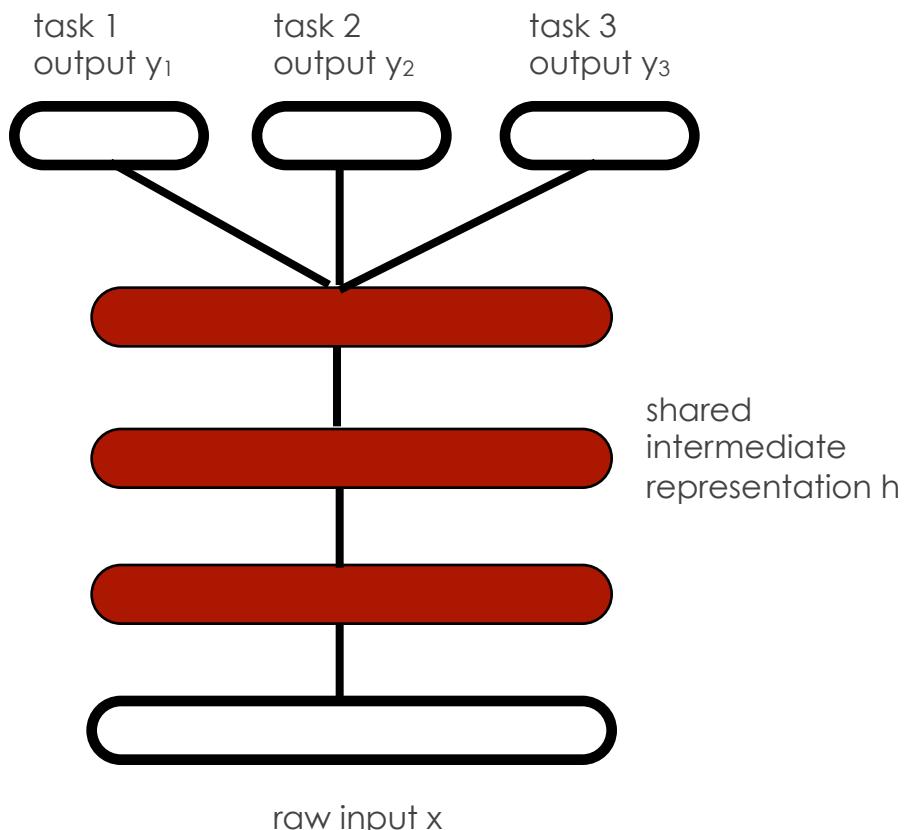
Deep Architecture in our Mind

- 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
- Want to learn / discover these concepts



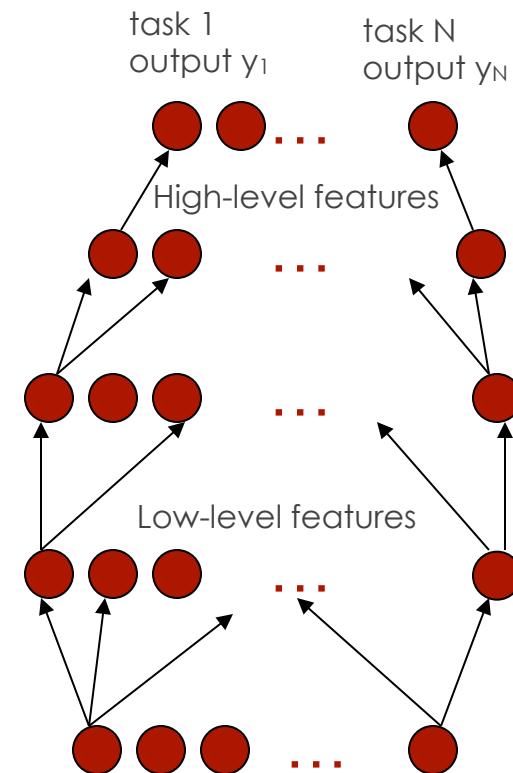
Deep Architectures and Sharing Statistical Strength, Multi-Task Learning

- 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

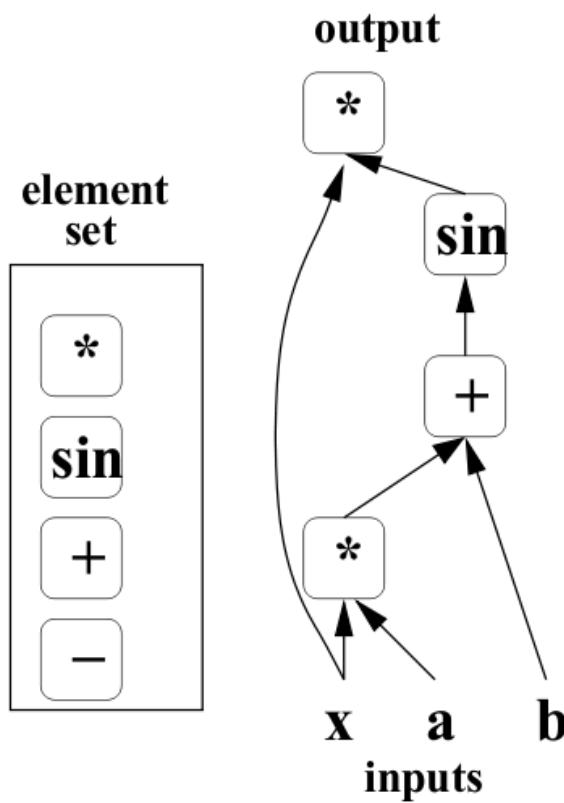


Feature and Sub-Feature Sharing

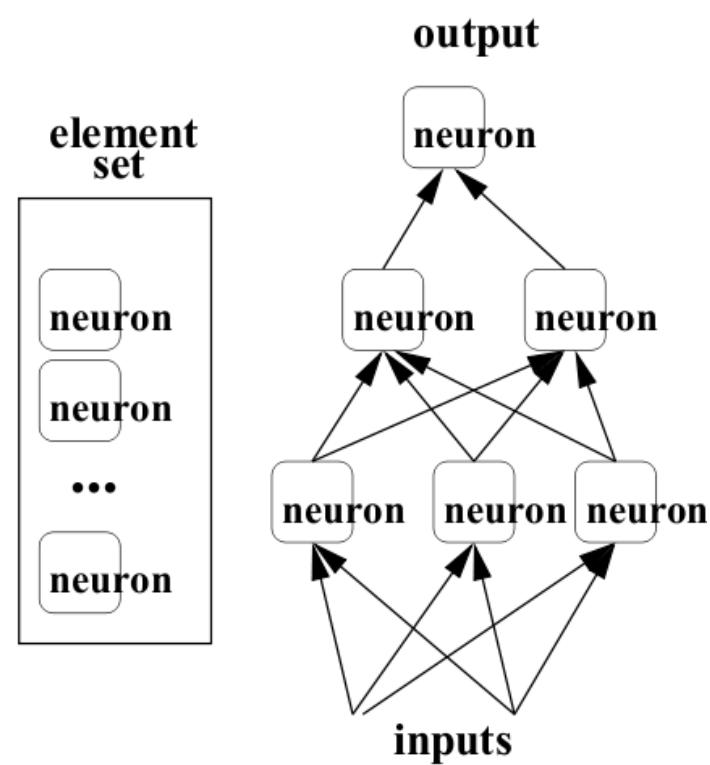
- Different tasks can share the same high-level feature
- Different high-level features can be built from the same set of lower-level features
- More levels = up to exponential gain in representational efficiency



Architecture Depth



Depth = 4



Depth = 3

Deep Architectures are More Expressive

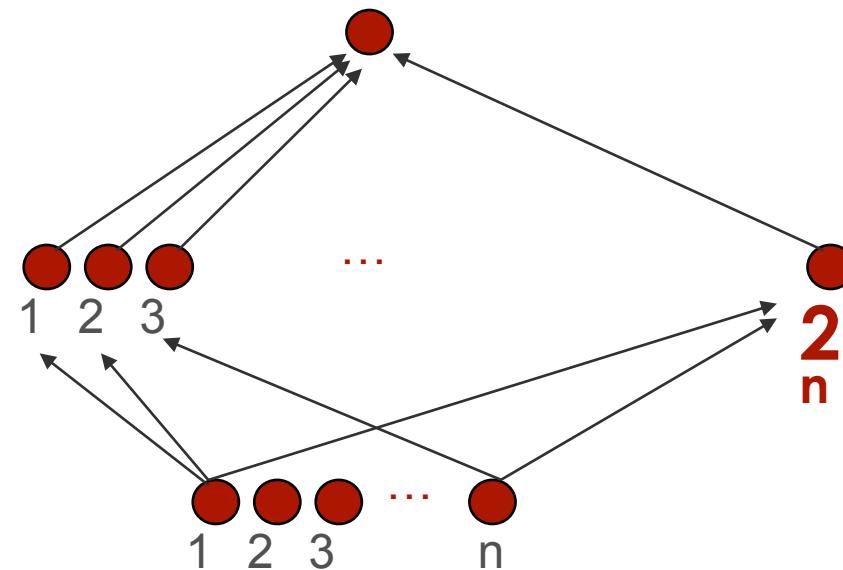
2 layers of

Logic gates
Formal neurons
RBF units

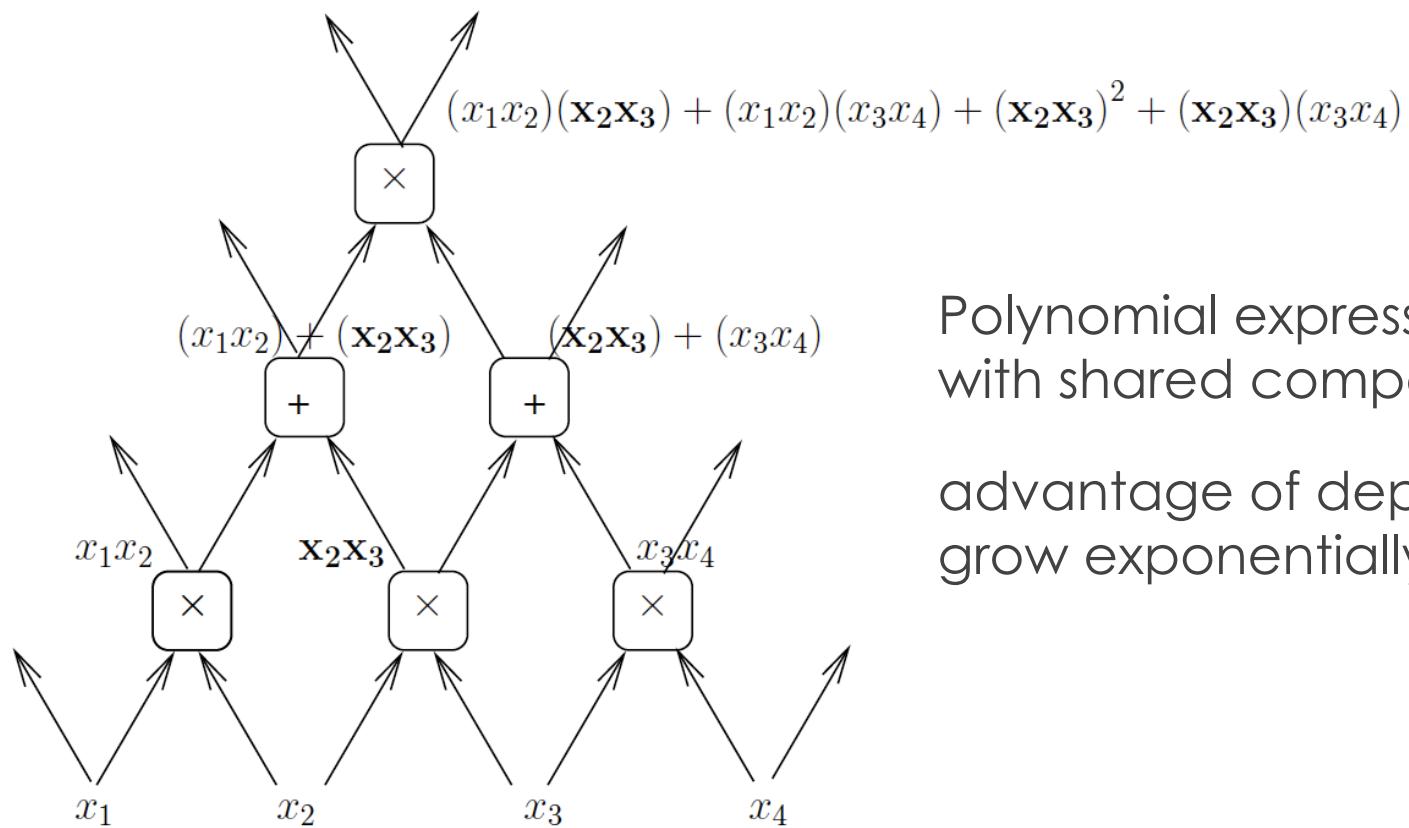
= universal approximator

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

Functions compactly represented with k layers may require exponential size with $k-1$ layers



Sharing Components in a Deep Architecture



Polynomial expressed
with shared components:
advantage of depth may
grow exponentially

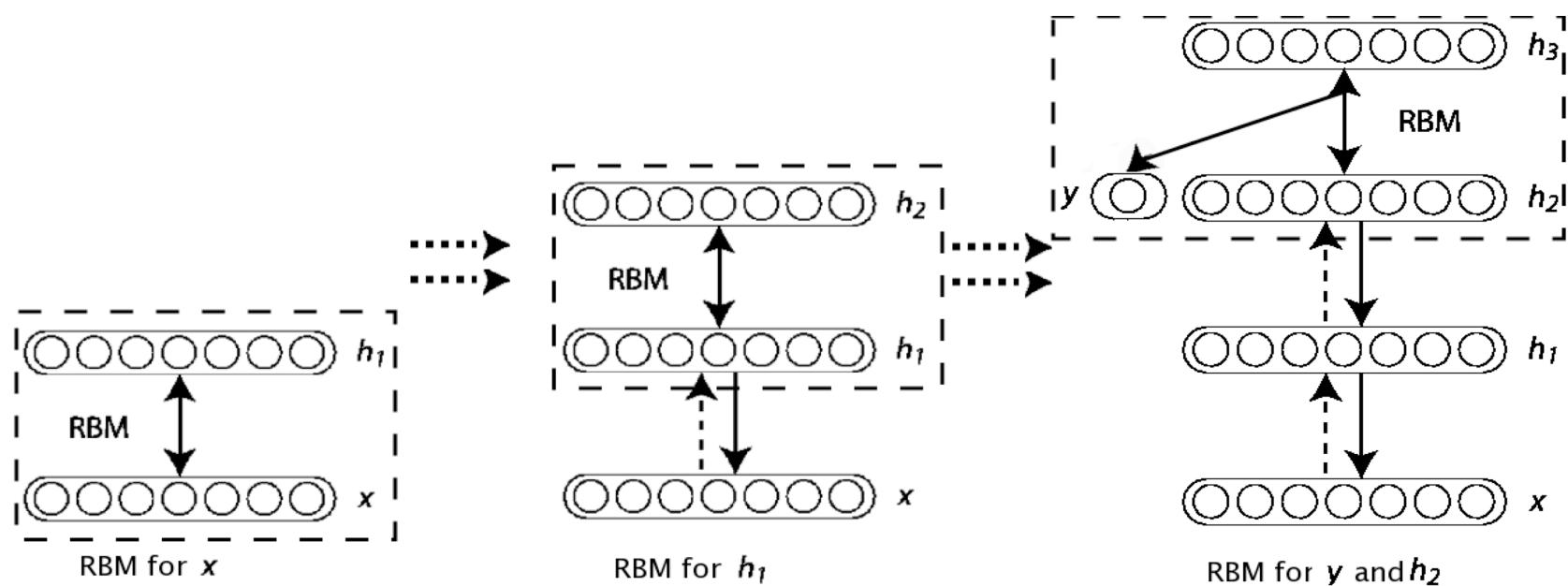
How to train Deep Architecture?

- Great expressive power of deep architectures
- How to train them?

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

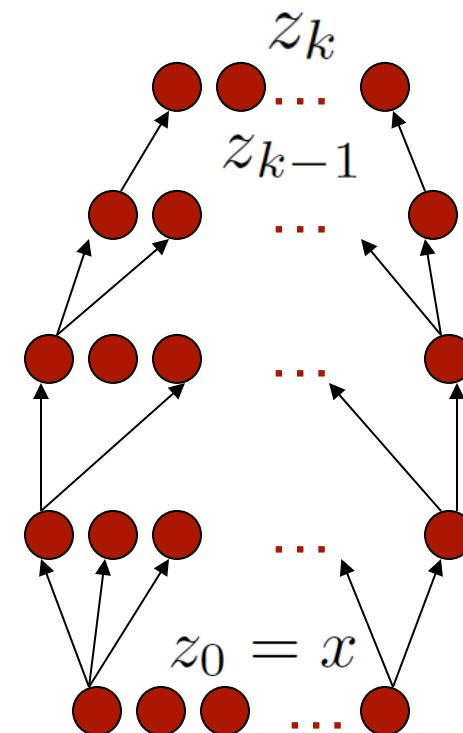
Greedy Layer-Wise Pre-Training



Stacking Restricted Boltzmann Machines (RBM) → Deep Belief Network (DBN)
→ Supervised deep neural network

Good Old Multi-Layer Neural Net

- Each layer outputs vector $z_k = \text{sigm}(b_k + W_k z_{k-1})$ from z_{k-1} of previous layer with params b_k (vector) and W_k (matrix).
- Output layer predicts parametrized distribution of target variable Y given input x



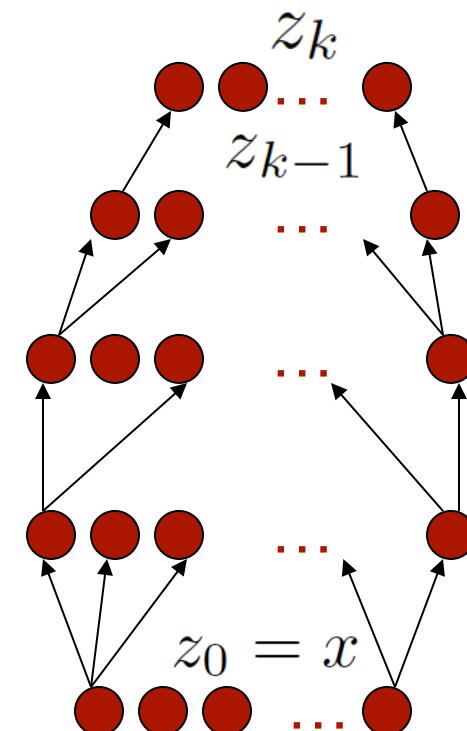
Training Multi-Layer Neural Nets

- Outputs: e.g. multinomial for multiclass classification with softmax output units

$$z_{ki} = \frac{e^{b_{ki} + W'_{ki} z_{k-1}}}{\sum_j e^{b_{kj} + W'_{kj} z_{k-1}}}$$

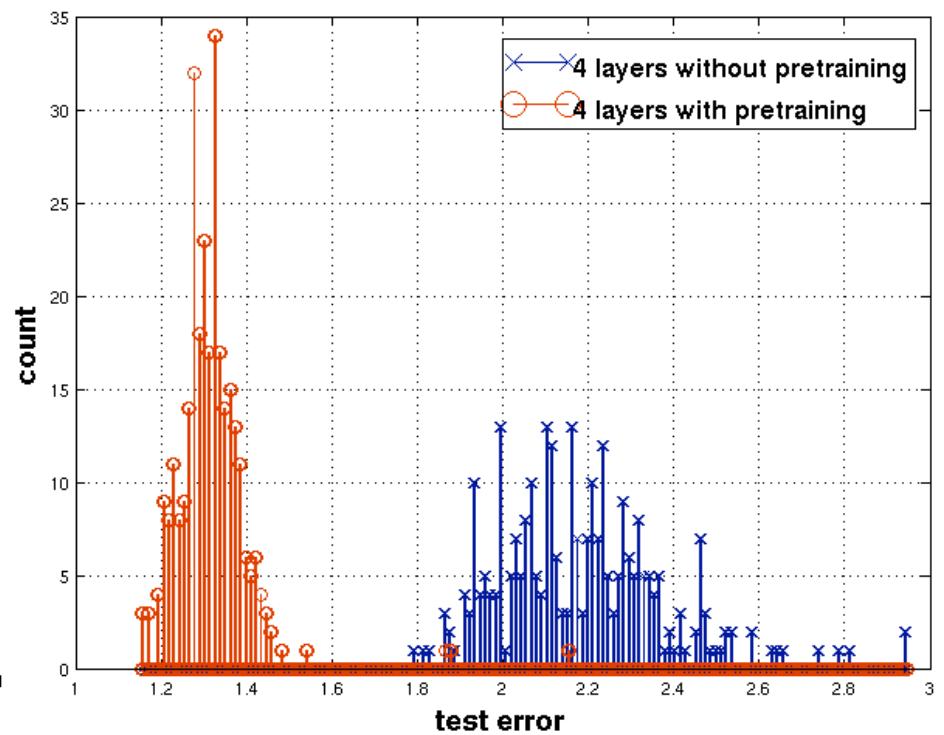
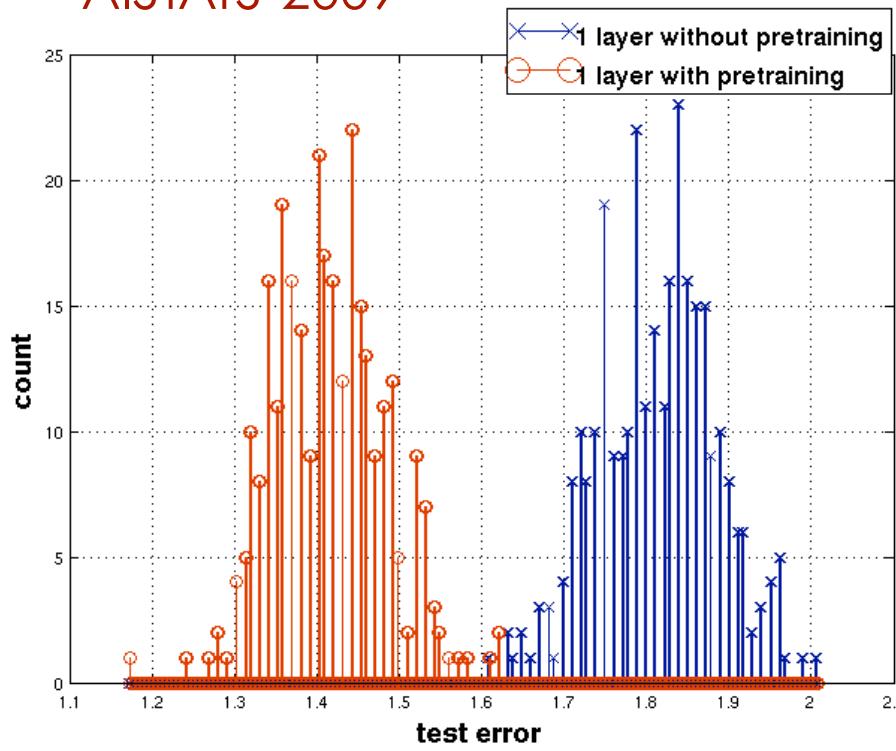
- Parameters are trained by gradient-based optimization of training criterion involving conditional log-likelihood, e.g.

$$-\log P(Y = y|x) = -\log z_{ky}$$



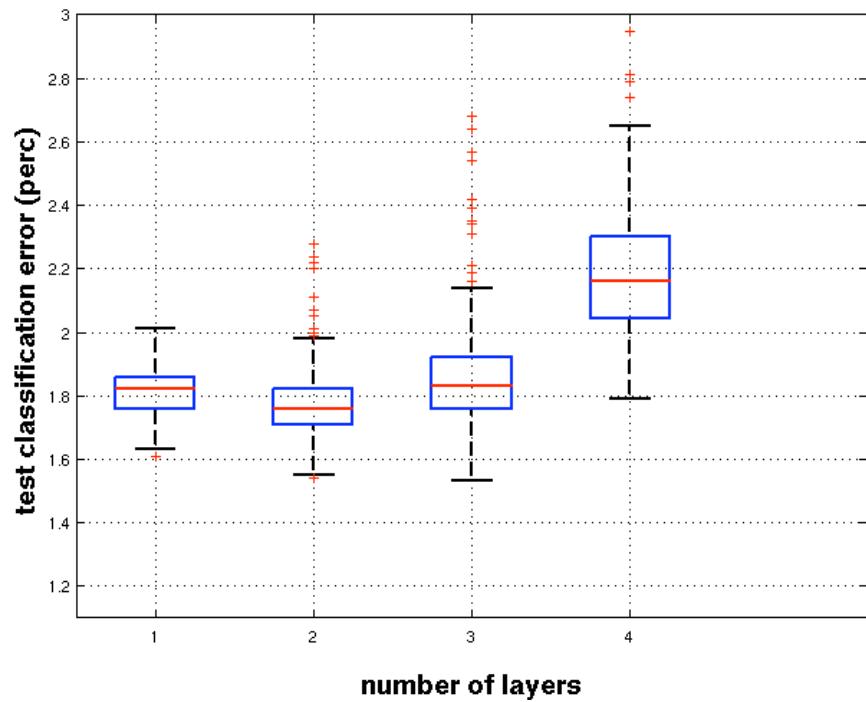
Effect of Unsupervised Pre-training

AISTATS'2009

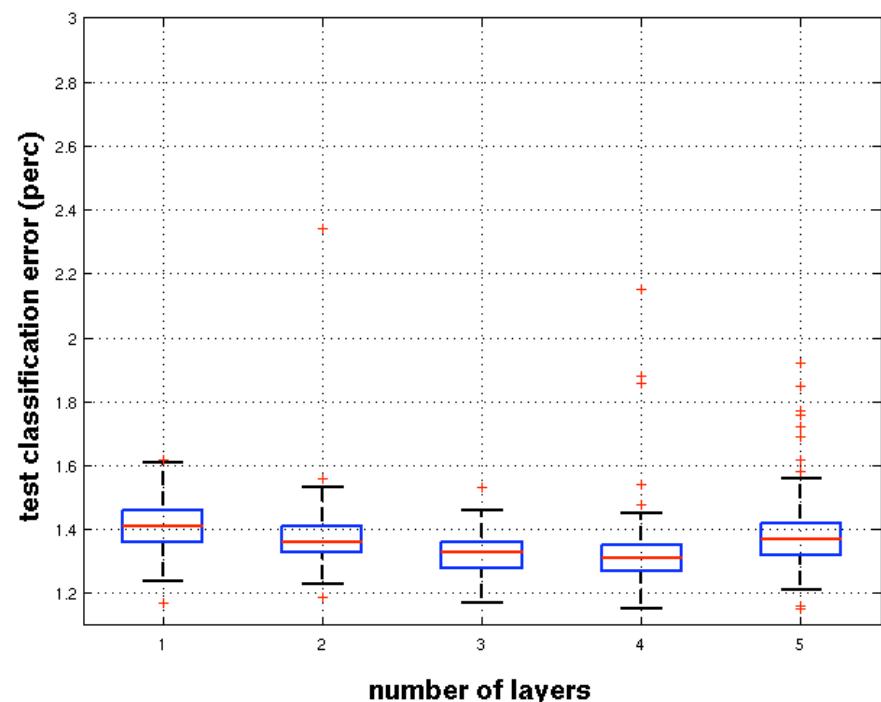


Effect of Depth

w/o pre-training



with pre-training

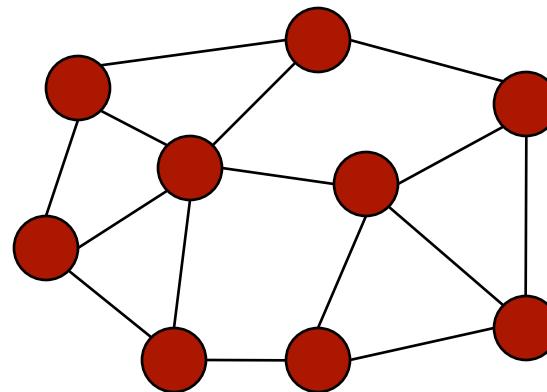


Boltzman Machines and MRFs

- Boltzmann machines: $P(x) = \frac{1}{Z} e^{-\text{Energy}(x)} = e^{c^T x + x^T W x}$
(Hinton 84)

- Markov Random Fields:

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

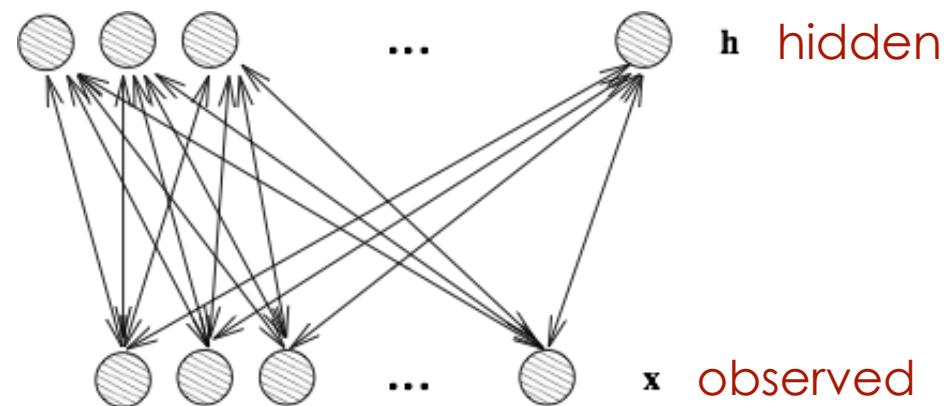


- More interesting with latent variables!

Restricted Boltzmann Machine

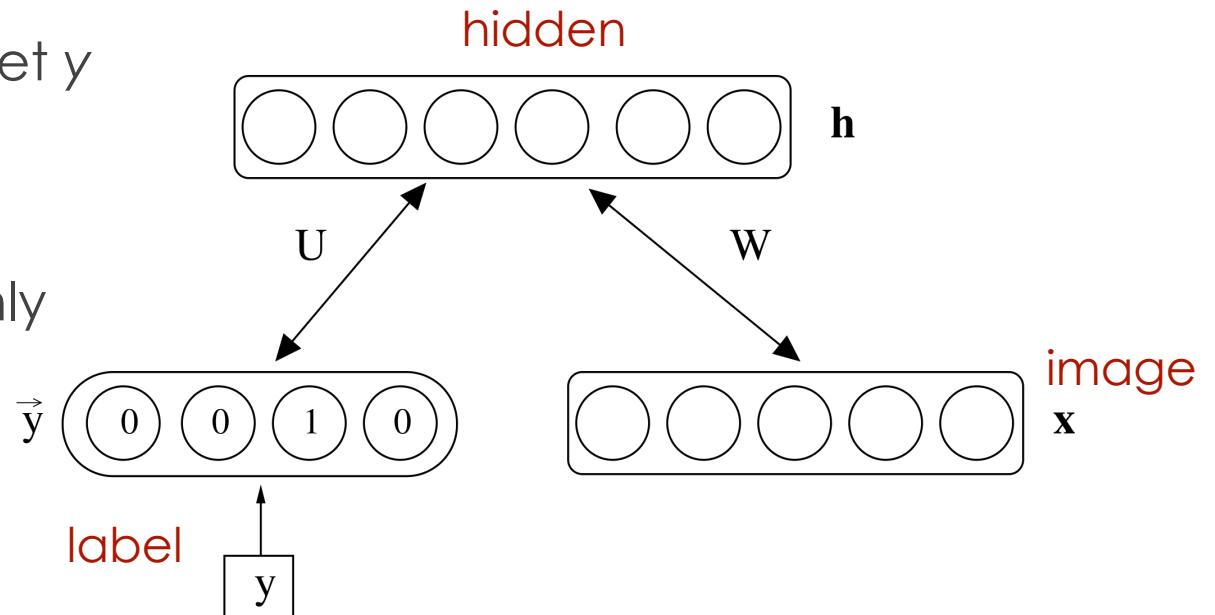
- The most popular building block for deep architectures
- Bipartite undirected graphical model

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



RBM with (image, label) visible units

- Can predict a subset y of the visible units given the others x
- Exactly if y takes only few values
- Gibbs sampling o/w



RBM^s are Universal Approximators

(LeRoux & Bengio 2008, Neural Comp.)

- 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 nb of hidden units = non-parametric
- Optimal training criterion for RBMs which will be stacked into a DBN is not the RBM likelihood

RBM Conditionals Factorize

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

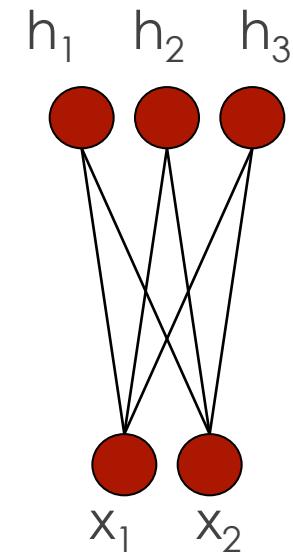
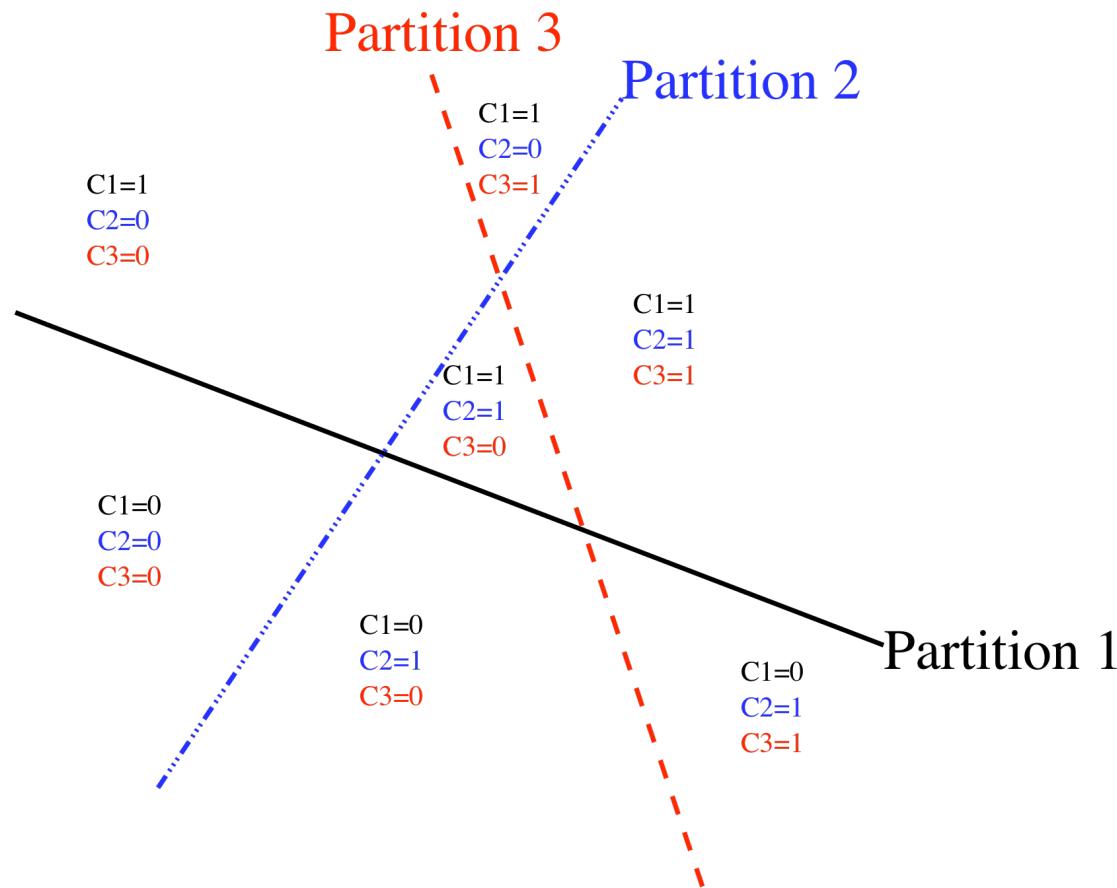
RBM Energy Gives Binomial Neurons

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

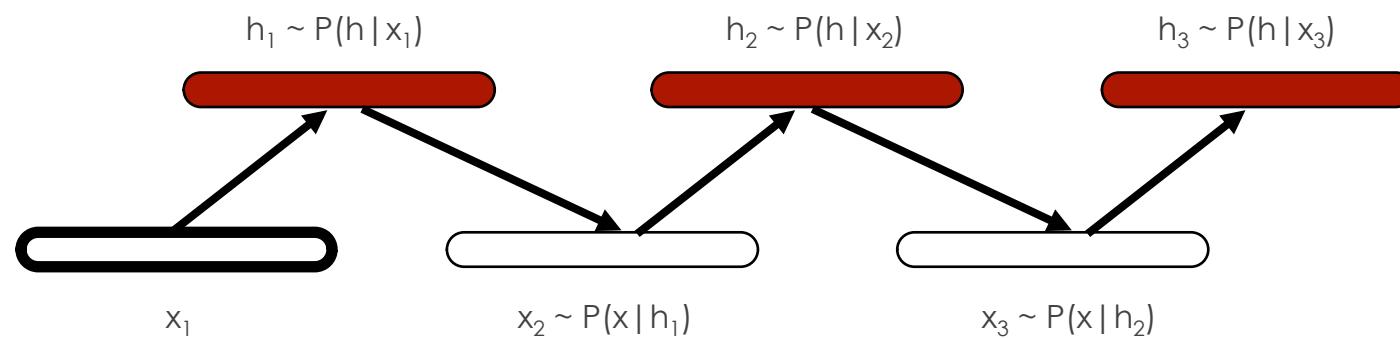
$$\begin{aligned} P(\mathbf{h}_i = 1 | \mathbf{x}) &= \frac{e^{1\mathbf{c}_i + 1W_i\mathbf{x} + \text{other terms}}}{e^{1\mathbf{c}_i + 1W_i\mathbf{x} + \text{other terms}} + e^{0\mathbf{c}_i + 0W_i\mathbf{x} + \text{other terms}}} \\ &= \frac{e^{\mathbf{c}_i + W_i\mathbf{x}}}{e^{\mathbf{c}_i + W_i\mathbf{x}}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

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

RBM Hidden Units Carve Input Space



Gibbs Sampling in RBMs



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

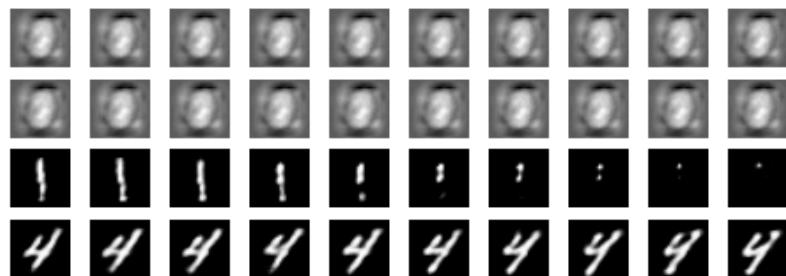
$$P(x, h) = \frac{1}{Z} e^{b^T h + c^T x + h^T W x}$$

- Easy inference
- Convenient Gibbs sampling
 $x \rightarrow h \rightarrow x \rightarrow h \dots$

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

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 $P(\mathbf{x})$ 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

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

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

Energy-Based Models Gradient

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

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

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

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

Training RBMs

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

Persistent CD: run negative Gibbs chain in
(PCD) background while weights slowly
change

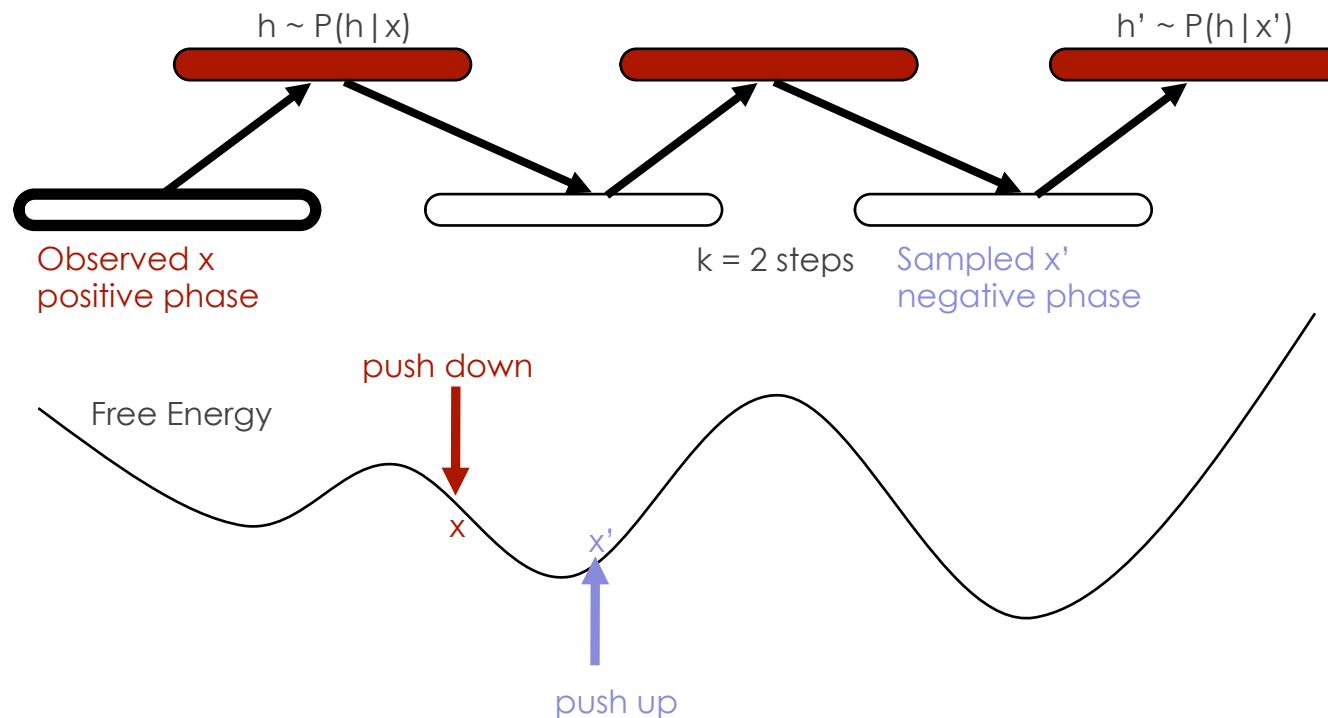
Fast PCD: two sets of weights, one with a large
learning rate only used for negative
phase, quickly exploring modes

Herding: Deterministic near-chaos dynamical
system defines both learning and sampling

Tempered MCMC: use higher temperature to escape
modes

Contrastive Divergence

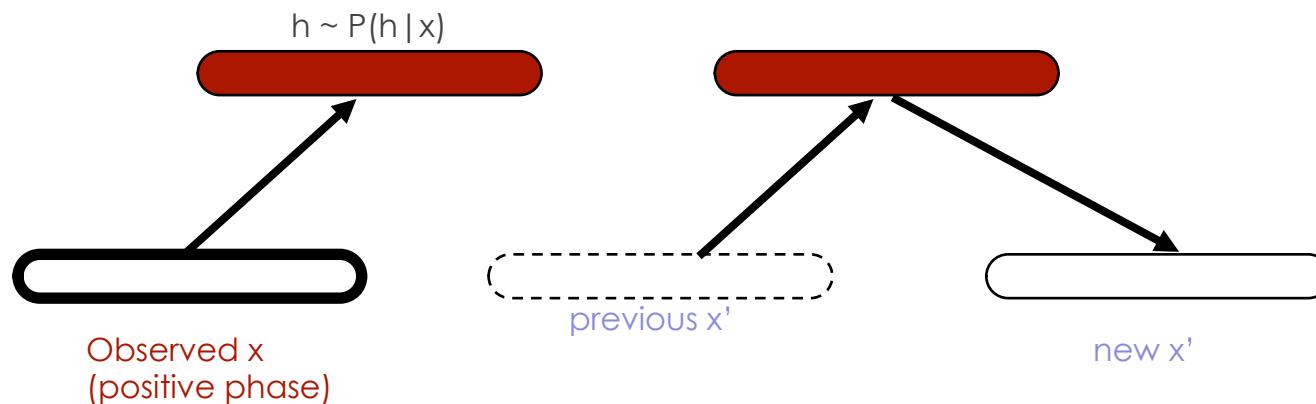
Contrastive Divergence (CD-k): start negative phase block Gibbs chain at observed x , run k Gibbs steps (Hinton 2002)



Persistent CD (PCD)

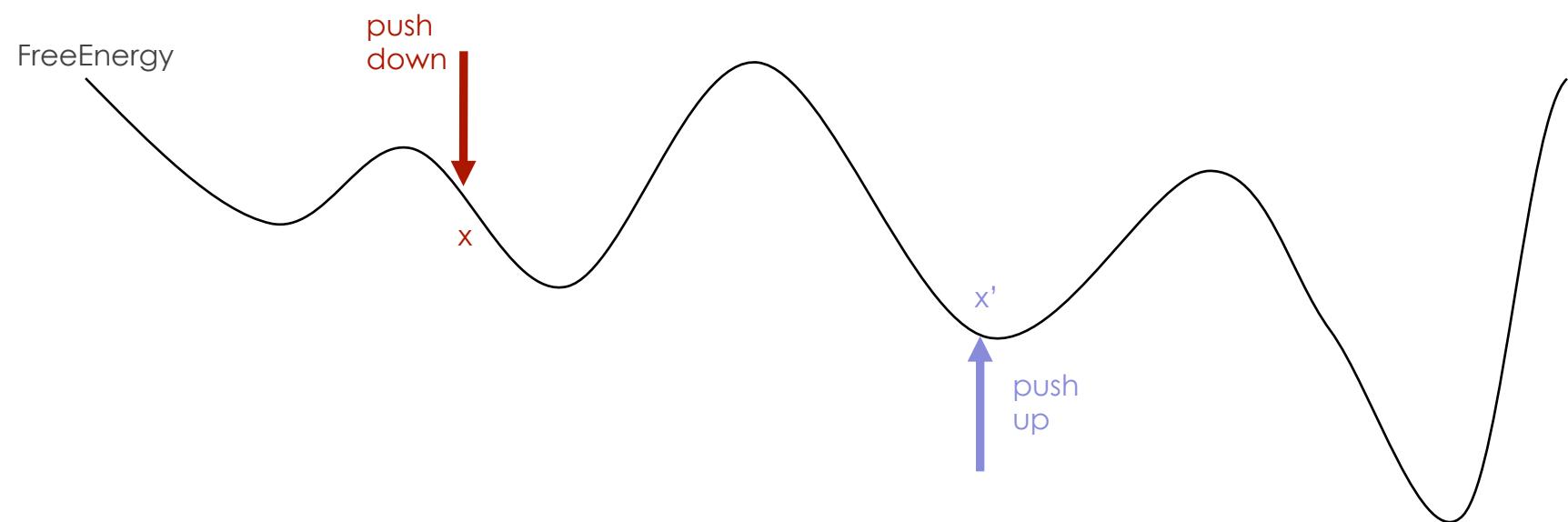
Run negative Gibbs chain in background while weights slowly change (Younes 2000, Tielemans 2008):

- 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



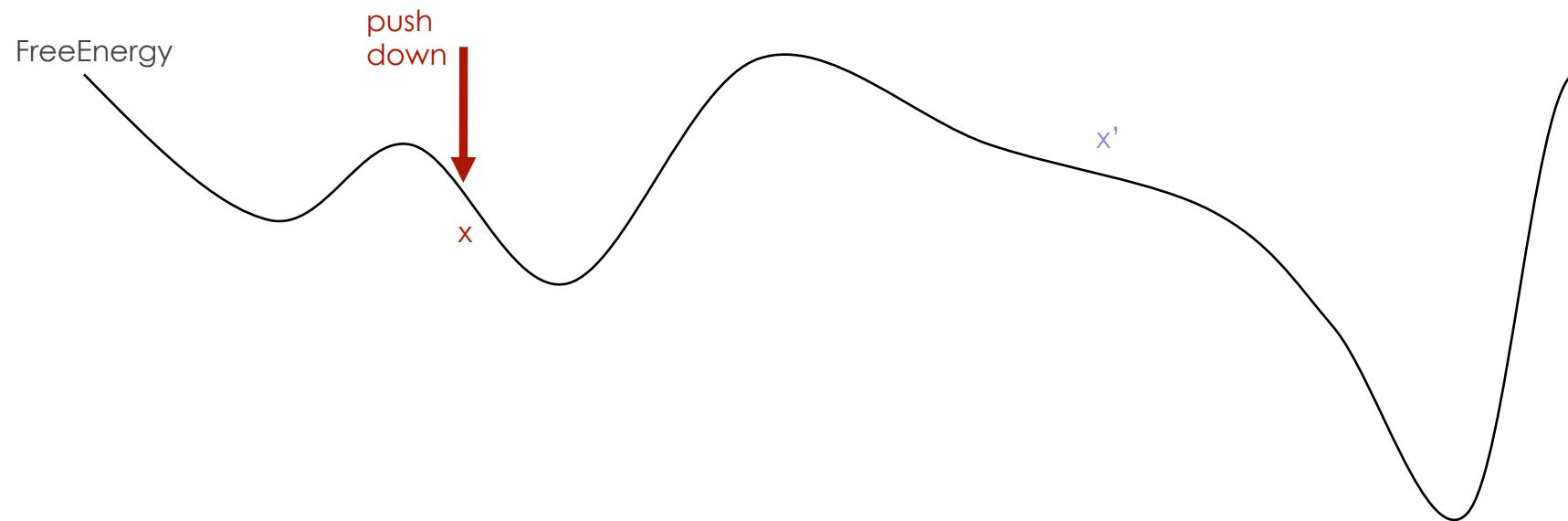
Persistent CD with large learning rate

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



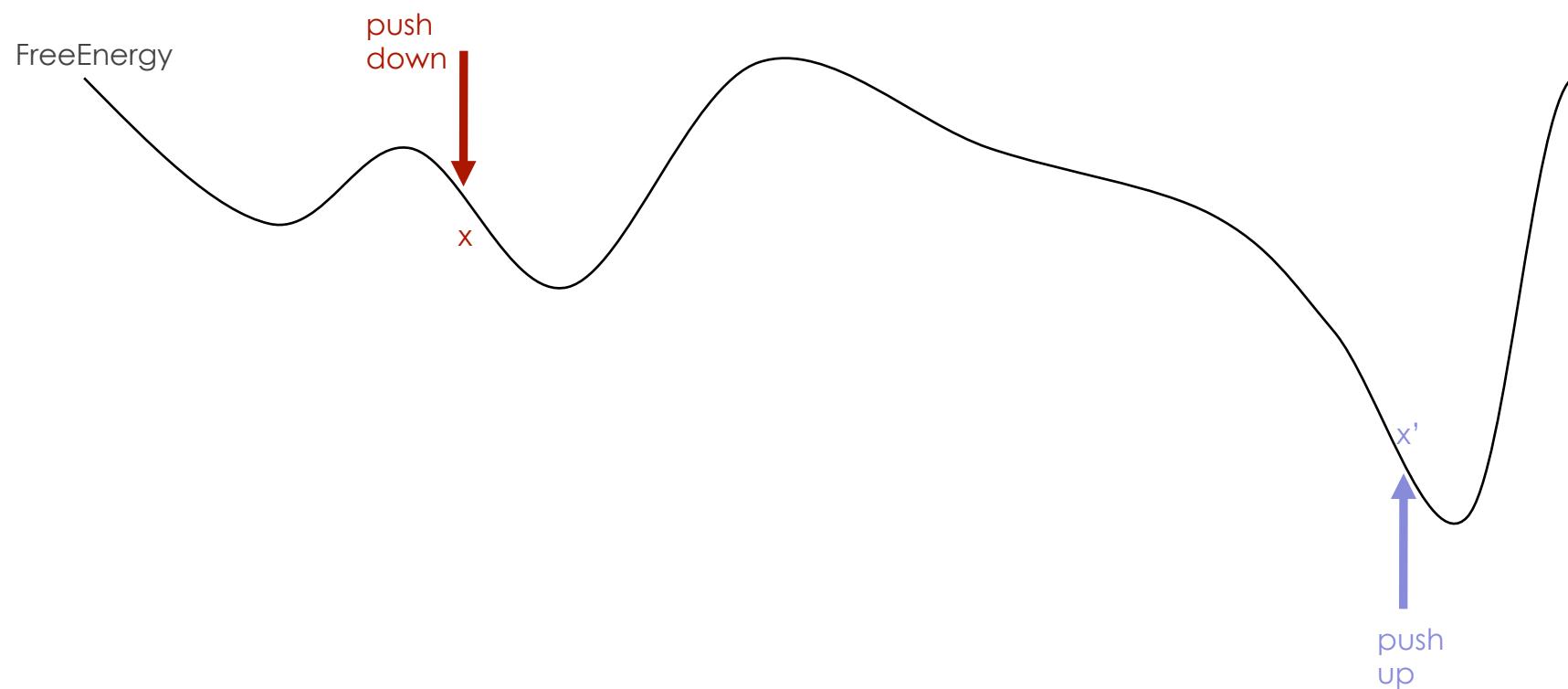
Persistent CD with large step size

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



Persistent CD with large learning rate

Negative phase samples quickly push up the energy of wherever 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

Herding MRFs

- Consider 0-temperature MRF with state s and weights w
- Fully observed case, observe values s^+ , dynamical system where s^- and W evolve
- Then statistics of samples s^- match the data's statistics, even if approximate max, as long as w remains bounded

$$\ell = E_{data\ s^+} [\sum_i w_i f_i(s^+)] - \max_s \sum_i w_i f_i(s)$$

$$s^- \leftarrow \operatorname{argmax}_s \sum_i w_i f_i(s)$$

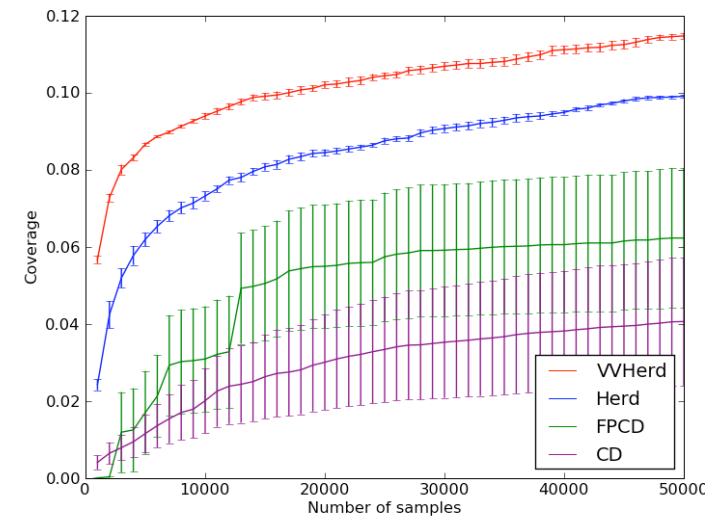
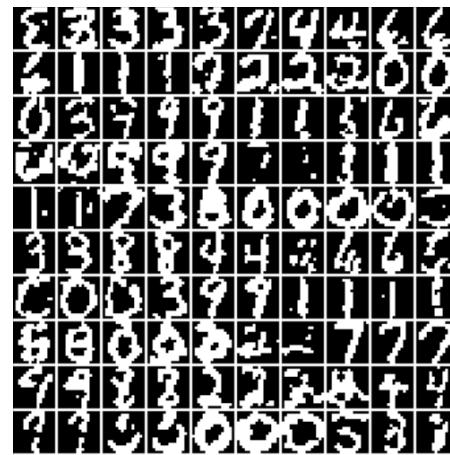
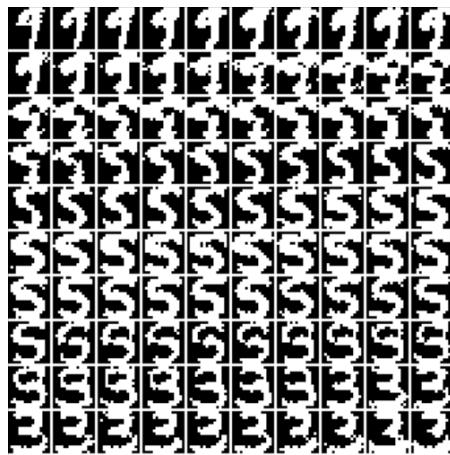
$$w \leftarrow w + E_{data\ s^+} [f(s^+)] - f(s^-)$$

$$E_{samples\ s^-} [f(s^-)] = E_{data\ s^+} [f(s^+)]$$

Herding RBMs

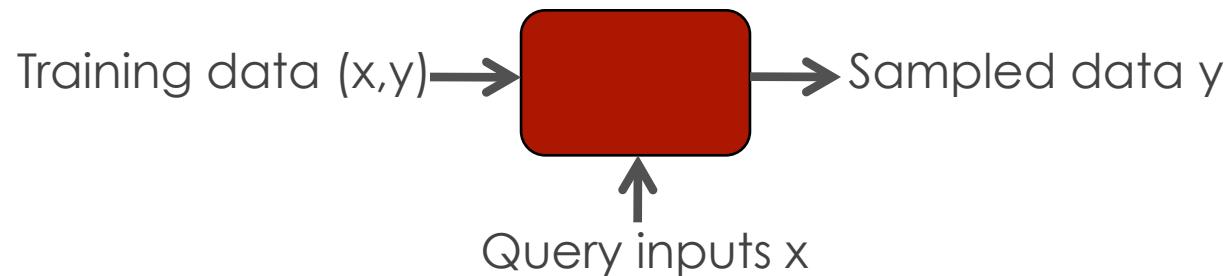
- Hidden part h of the state $s = (x, h)$
- Binomial state variables $s_i \in \{-1, 1\}$
- Statistics f $s_i, s_i s_j$
- Optimize h given x in positive phase $\ell = E_{data \ x^+} [\max_h \sum_i w_i f_i(x^+, h)] - \max_s \sum_i w_i f_i(s)$
$$s^- \leftarrow \operatorname{argmax}_s \sum_i w_i f_i(s)$$
$$w \leftarrow w + E_{data \ x^+} [\max_h f(x^+, h)] - f(s^-)$$
- In practice, greedy maximization works, exploiting RBM structure

Fast Mixing with Herding



The Sampler as a Generative Model

- Instead of the traditional clean separation between model and sampling procedure
- Consider the overall effect of combining some adaptive procedure with a sampling procedure **as the generative model**
- Can be evaluated as such
(without reference to some underlying probability model)



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

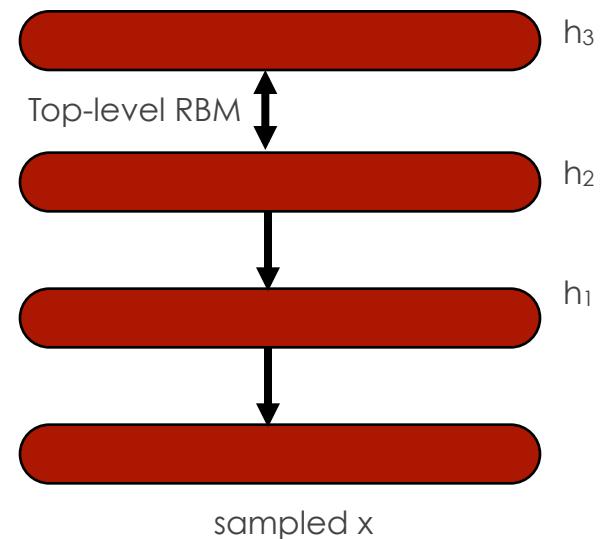
| Training Procedure | TMCMC | Gibbs (random start) | Gibbs (test start) |
|--------------------|-------------------|----------------------|--------------------|
| TMCMC | 215.45 ± 2.24 | 88.43 ± 2.75 | 60.04 ± 2.88 |
| PCD | 44.70 ± 2.51 | -28.66 ± 3.28 | -175.08 ± 2.99 |
| CD | -2165 ± 0.53 | -2154 ± 0.63 | -842.76 ± 6.17 |

Deep Belief Networks

- DBN = sigmoidal belief net with RBM joint for top two layers

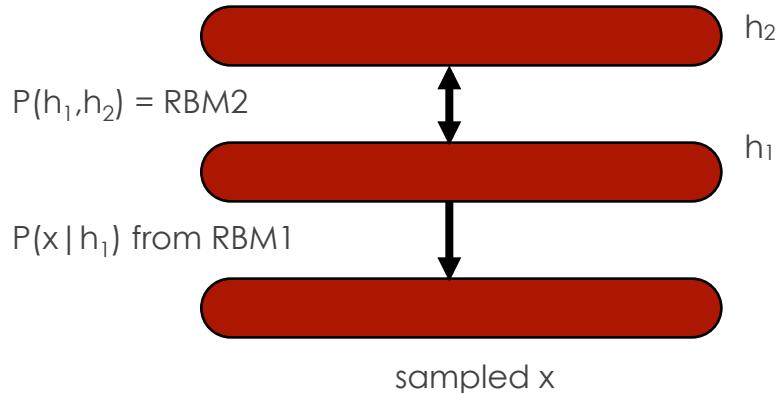
$$P(\mathbf{x}, \mathbf{h}^1, \dots, \mathbf{h}^\ell) = P(\mathbf{h}^{\ell-1}, \mathbf{h}^\ell) \left(\prod_{k=1}^{\ell-2} P(\mathbf{h}^k | \mathbf{h}^{k+1}) \right) P(\mathbf{x} | \mathbf{h}^1)$$

- Sampling:
 - Sample from top RBM
 - Sample from level k given k+1
- Level k given level k+1 = same parametrization as RBM conditional: stacking RBMs \rightarrow DBN



From RBM to DBN

- RBM specifies $P(v, h)$ from $P(v | h)$ and $P(h | v)$
- Implicitly defines $P(v)$ and $P(h)$
- Keep $P(v | h)$ from 1st RBM and replace $P(h)$ by the distribution generated by 2nd level RBM



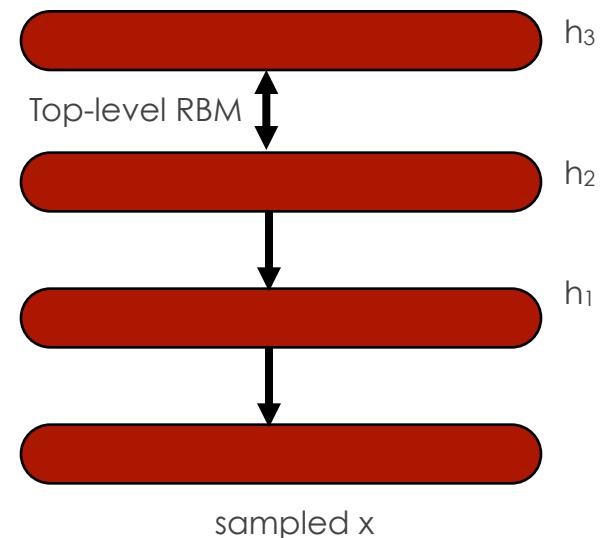
Deep Belief Networks

- Easy approximate inference
 - $P(h_{k+1} | h_k)$ approximated from the associated RBM
 - Approximation because $P(h_{k+1})$ differs between RBM and DBN

- Training:
 - Variational bound justifies greedy layerwise training of RBMs

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

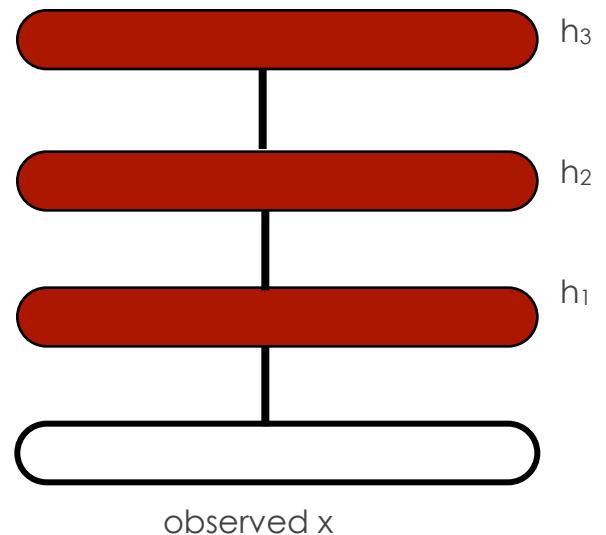
- How to train all levels together?



Deep Boltzman 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

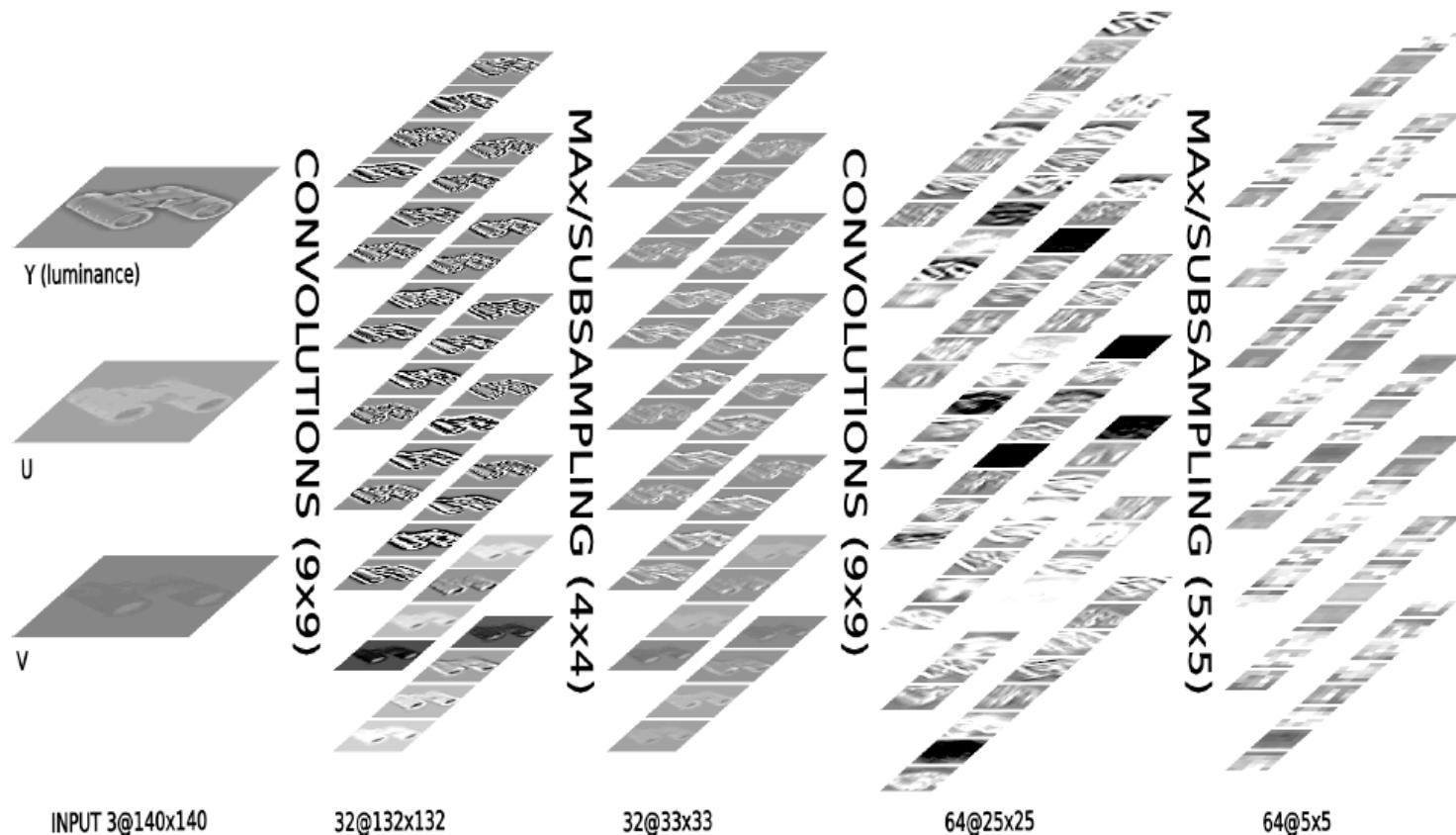


Estimating Log-Likelihood

- 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

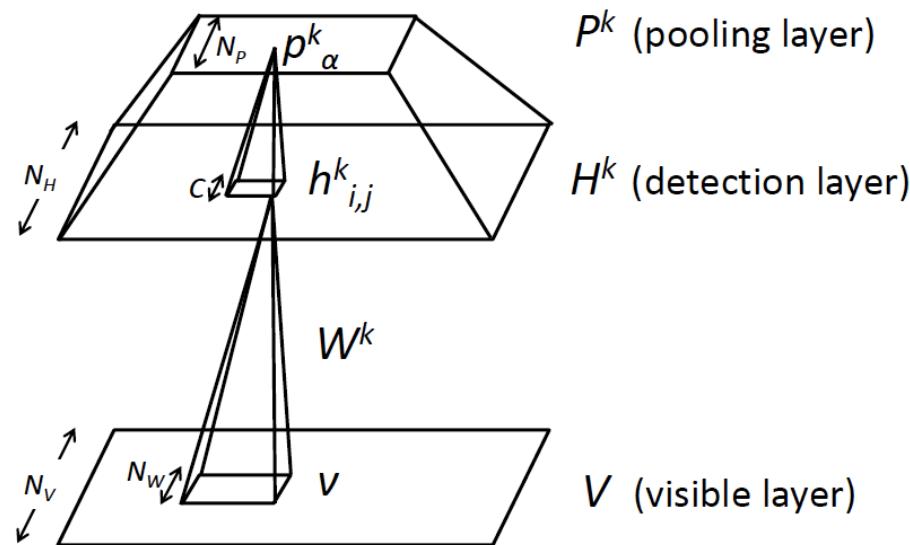
Deep Convolutional Architectures

Mostly from Le Cun's group (NYU), also Ng (Stanford):
state-of-the-art on MNIST digits, Caltech-101 objects, faces

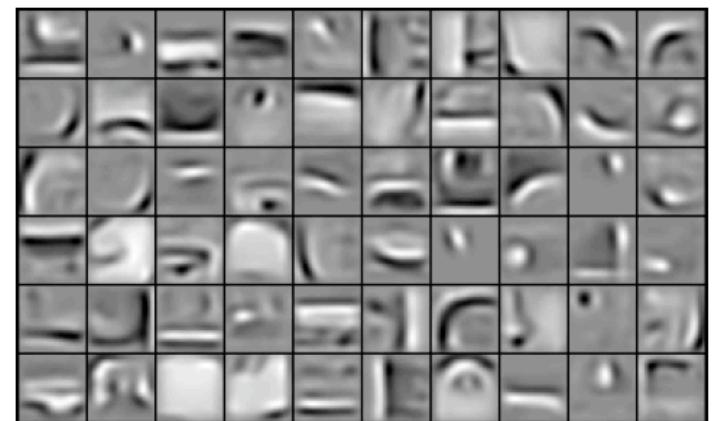


Convolutional DBNs

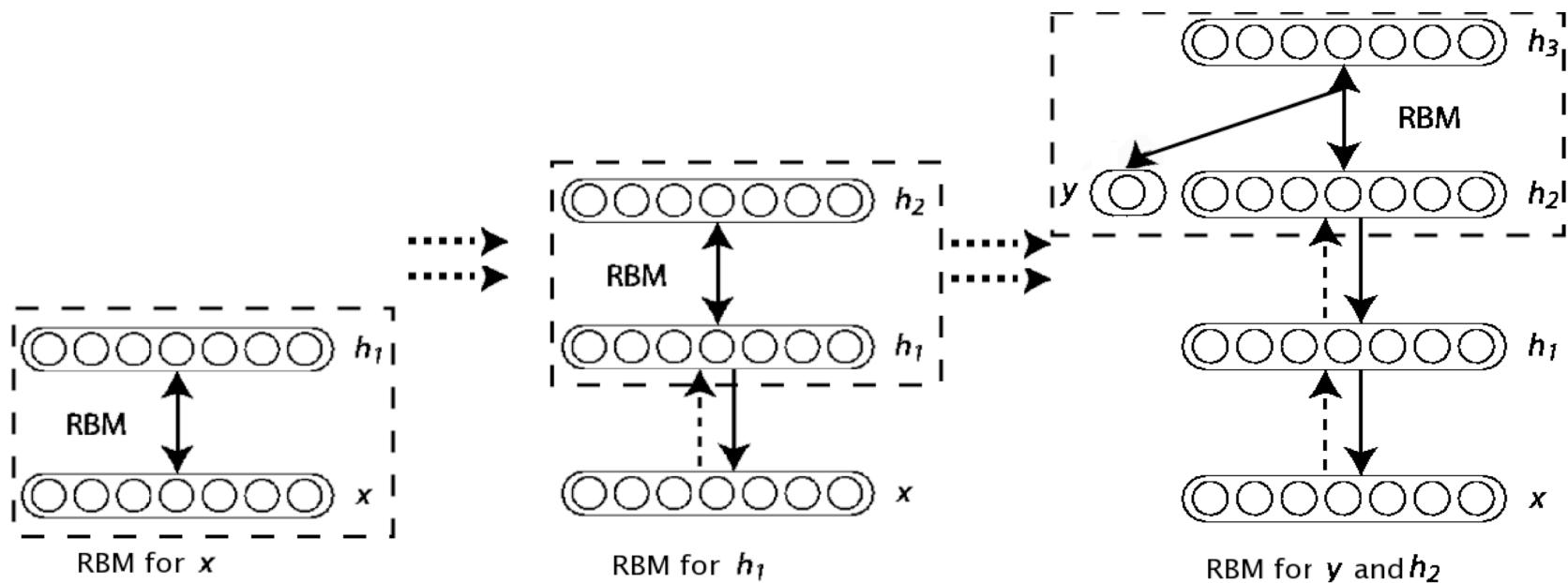
(Lee et al, ICML'2009)



faces, cars, airplanes, motorbikes



Back to Greedy Layer-Wise Pre-Training

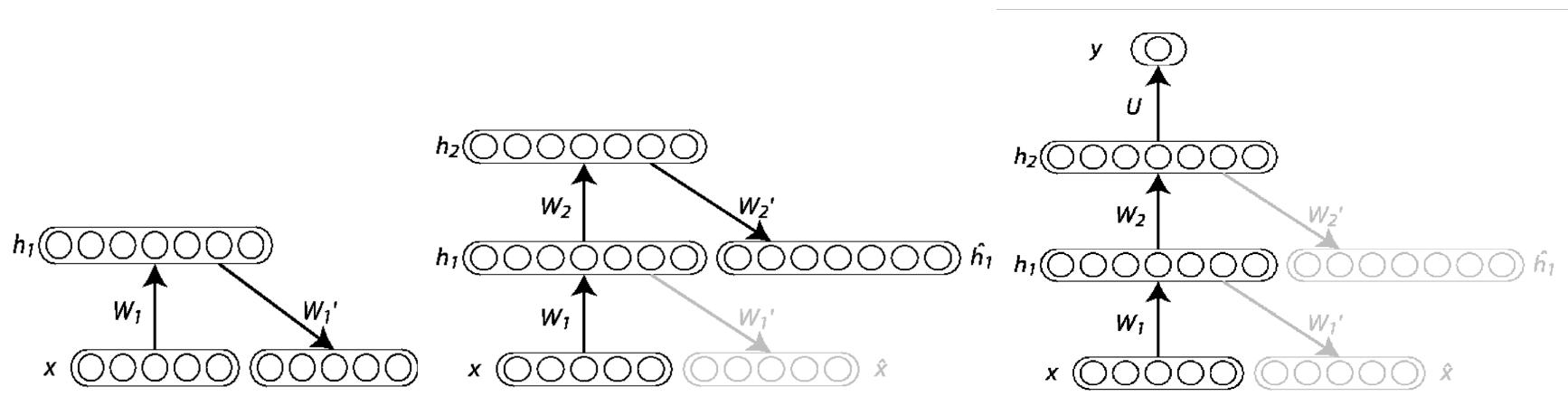


Stacking Restricted Boltzmann Machines (RBM) → Deep Belief Network (DBN)
→ Supervised deep neural network

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



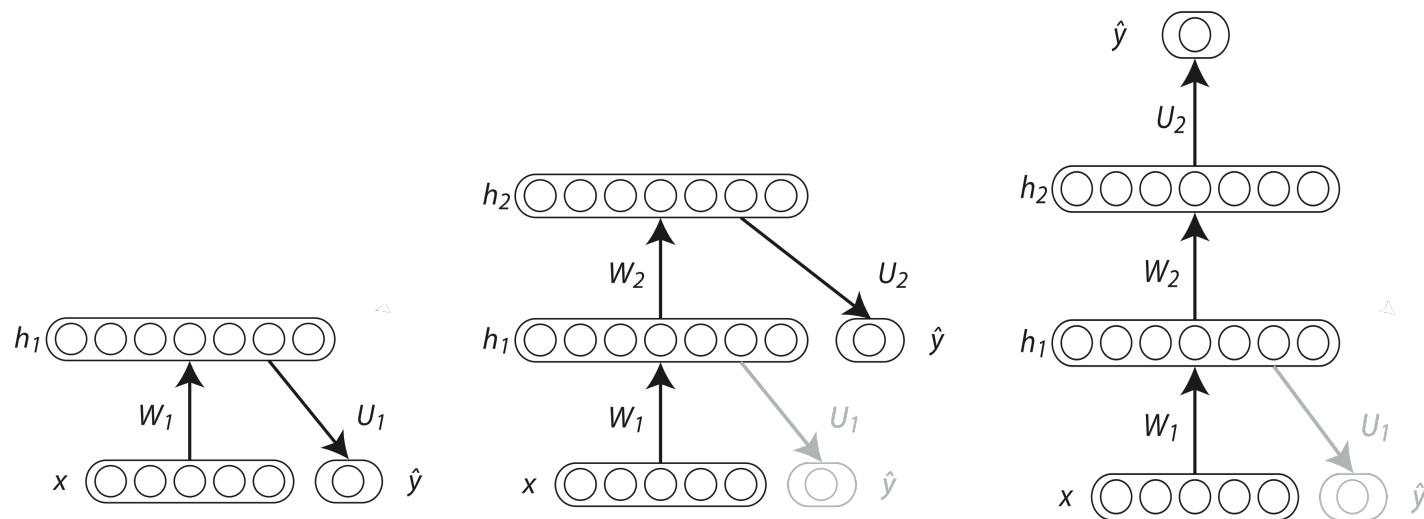
Greedy layer-wise unsupervised pre-training also works with auto-encoders

Auto-encoders and CD

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

$$\begin{aligned}\frac{\partial \log P(x_1)}{\partial \theta} &= \sum_{s=1}^{t-1} \left(E\left[\frac{\partial \log P(x_s|h_s)}{\partial \theta} \middle| x_1 \right] + E\left[\frac{\partial \log P(h_s|x_{s+1})}{\partial \theta} \middle| x_1 \right] \right) \\ &+ E\left[\frac{\partial \log P(x_t)}{\partial \theta} \middle| x_1 \right]\end{aligned}\quad (\text{Bengio \& Delalleau 2009})$$

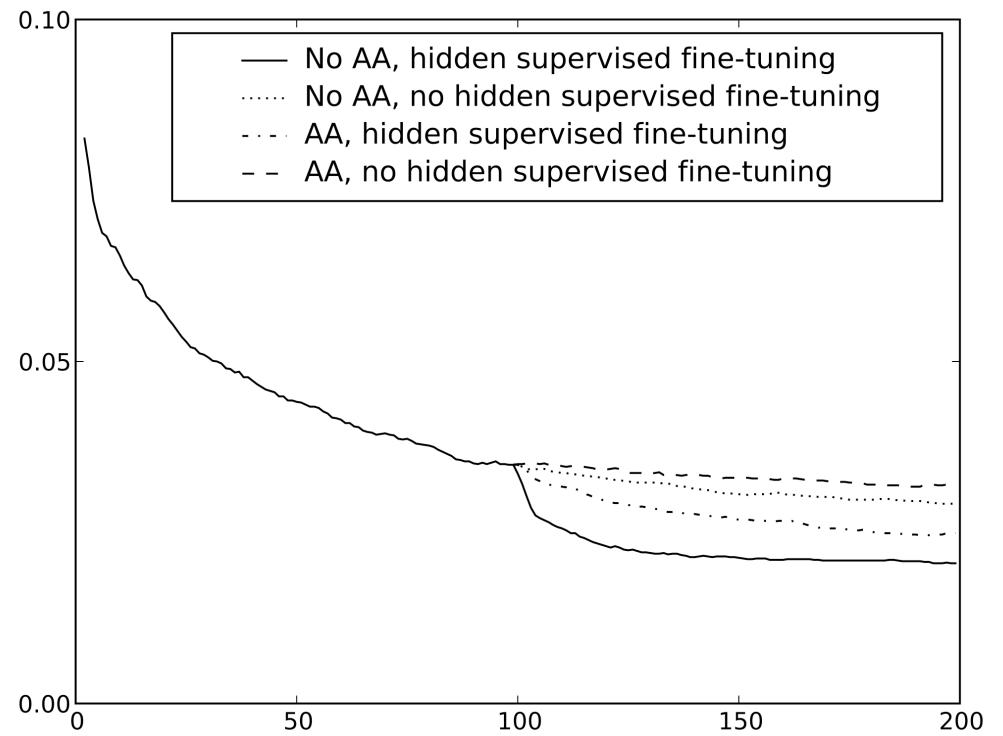
Greedy Layerwise Supervised Training



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



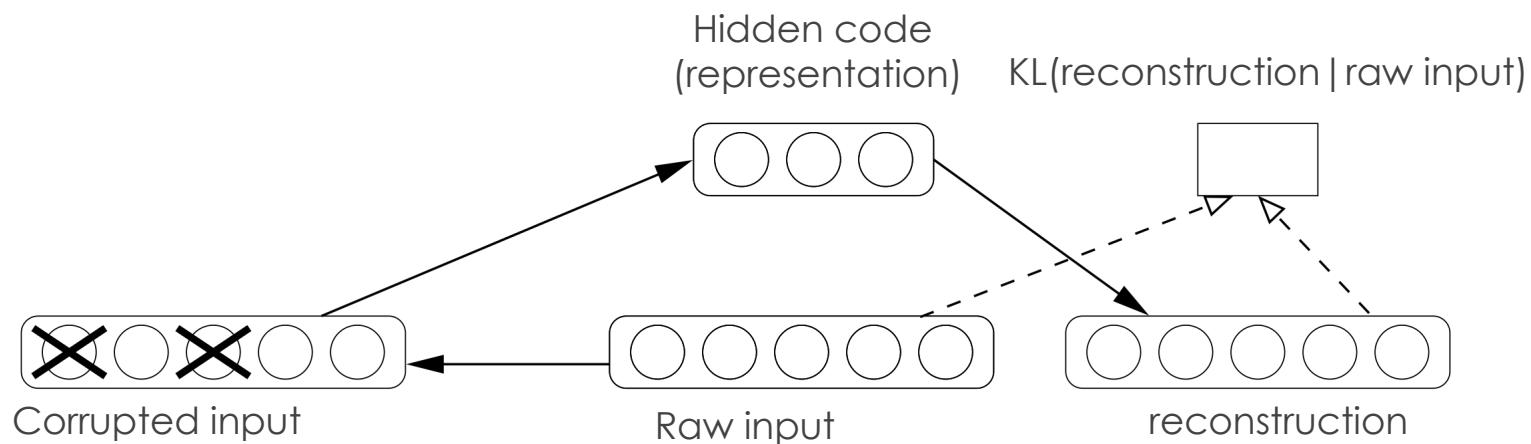
Sparse Auto-Encoders

(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

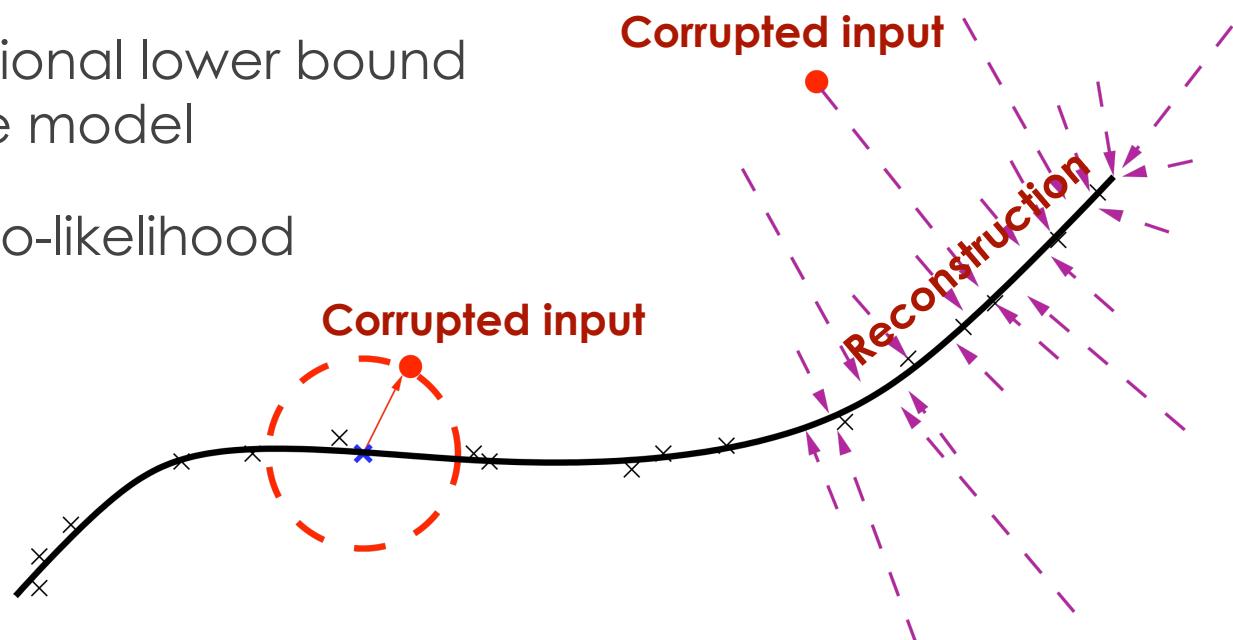
(Vincent et al, 2008)



- Corrupt the input (e.g. set 25% of inputs to 0)
- Reconstruct the uncorrupted input
- Use uncorrupted encoding as input to next level

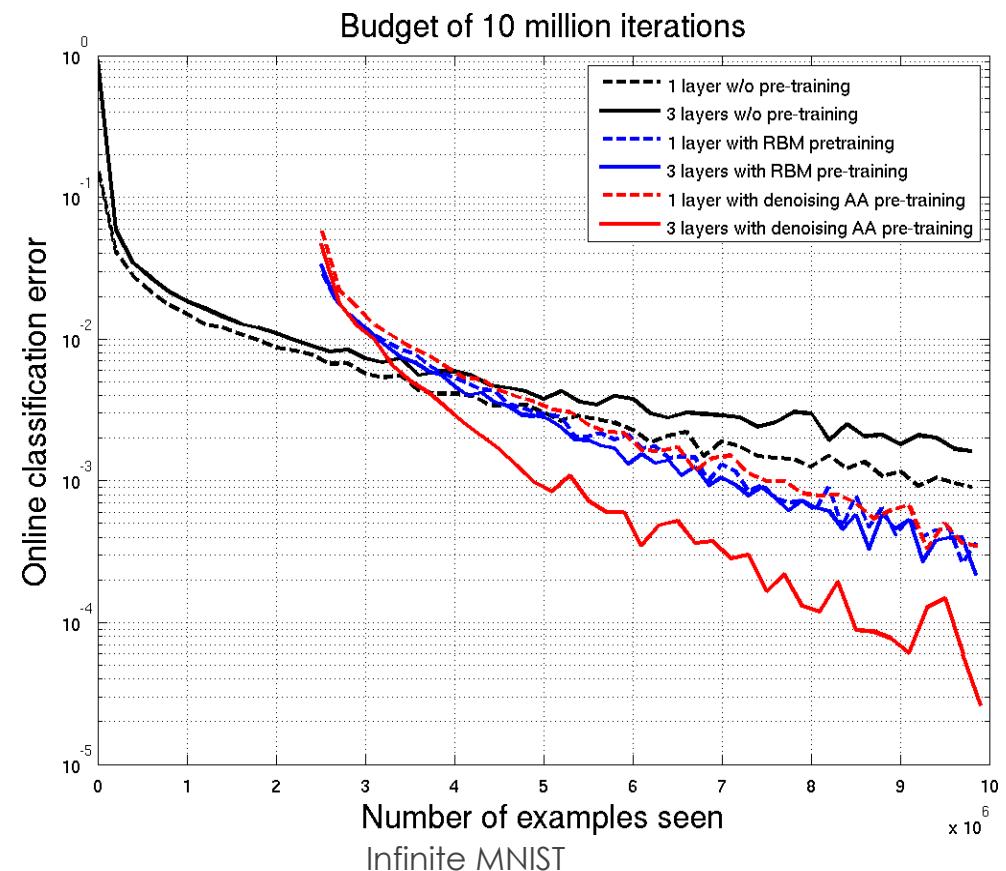
Denoising Auto-Encoder

- 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 unsupervised pre-training
- Generative model is semi-parametric



Denoising Auto-Encoders: Benchmarks

| | |
|--|---|
| basic: subset of MNIST digits. | (10 000 training samples) |
| rot: applied random rotation (angle between 0 and 2π radians) |  |
| bg-rand: background made of random pixels (value in 0...255) |  |
| bg-img: background is random patch from one of 20 images |  |
| rot-bg-img: combination of rotation and background image |  |
| rect: discriminate between tall and wide rectangles. |  |
| rect-img: same but rectangles are random image patches |  |
| convex: discriminate between convex and non-convex shapes. |  |

Denoising Auto-Encoders: Results

| Problem | SVM _{rbf} | DBN-1 | DBN-3 | SAA-3 | <u>SdA-3 (ν)</u> | SVM _{rbf} (ν) |
|------------|--|---|---|---|---|------------------------------|
| basic | 3.03 _{± 0.15} | 3.94 ± 0.17 | 3.11 ± 0.15 | 3.46 ± 0.16 | 2.80 _{± 0.14} (10%) | 3.07 (10%) |
| rot | 11.11 ± 0.28 | 14.69 ± 0.31 | 10.30 _{± 0.27} | 10.30 _{± 0.27} | 10.29 _{± 0.27} (10%) | 11.62 (10%) |
| bg-rand | 14.58 ± 0.31 | 9.80 ± 0.26 | 6.73 _{± 0.22} | 11.28 ± 0.28 | 10.38 ± 0.27 (40%) | 15.63 (25%) |
| bg-img | 22.61 ± 0.37 | 16.15 _{± 0.32} | 16.31 _{± 0.32} | 23.00 ± 0.37 | 16.68 _{± 0.33} (25%) | 23.15 (25%) |
| rot-bg-img | 55.18 ± 0.44 | 52.21 ± 0.44 | 47.39 ± 0.44 | 51.93 ± 0.44 | 44.49 _{± 0.44} (25%) | 54.16 (10%) |
| rect | 2.15 _{± 0.13} | 4.71 ± 0.19 | 2.60 ± 0.14 | 2.41 ± 0.13 | 1.99 _{± 0.12} (10%) | 2.45 (25%) |
| rect-img | 24.04 ± 0.37 | 23.69 ± 0.37 | 22.50 ± 0.37 | 24.05 ± 0.37 | 21.59 _{± 0.36} (25%) | 23.00 (10%) |
| convex | 19.13 ± 0.34 | 19.92 ± 0.35 | 18.63 _{± 0.34} | 18.41 _{± 0.34} | 19.06 _{± 0.34} (10%) | 24.20 (10%) |

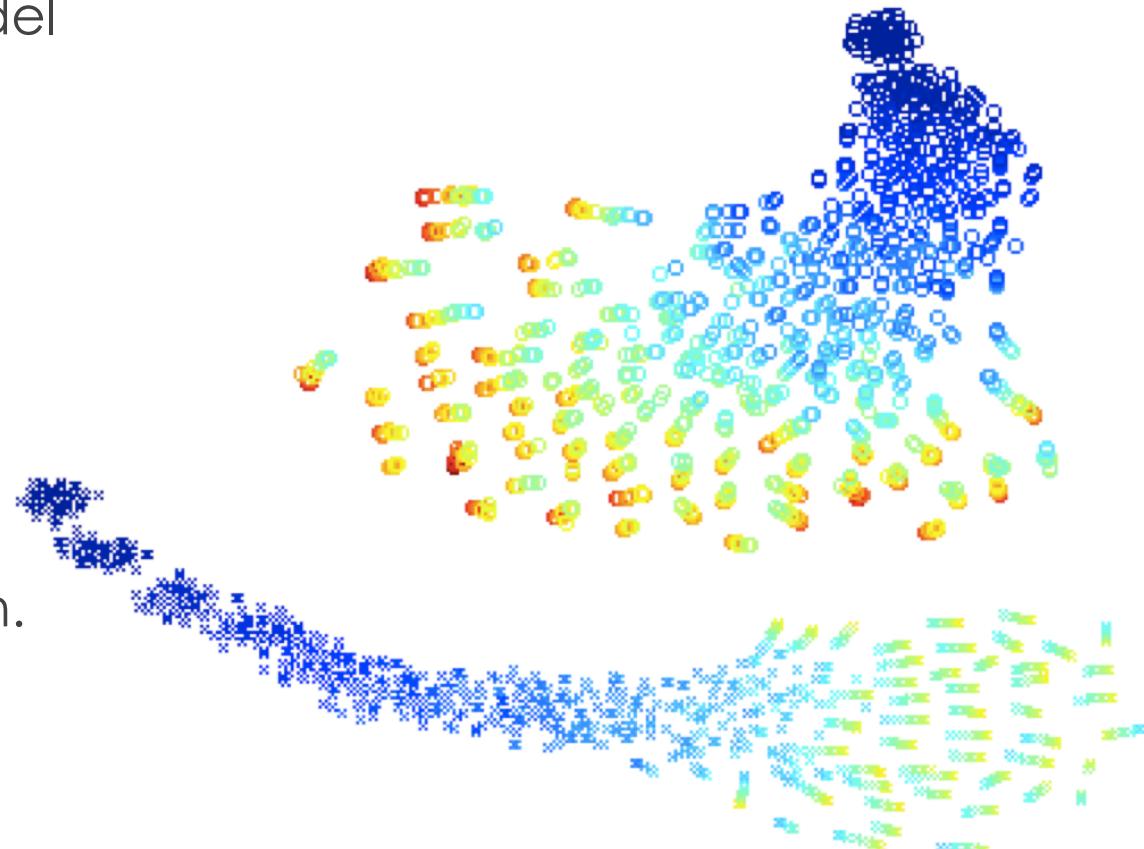
Why is Unsupervised Pre-Training Working So Well?

- **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

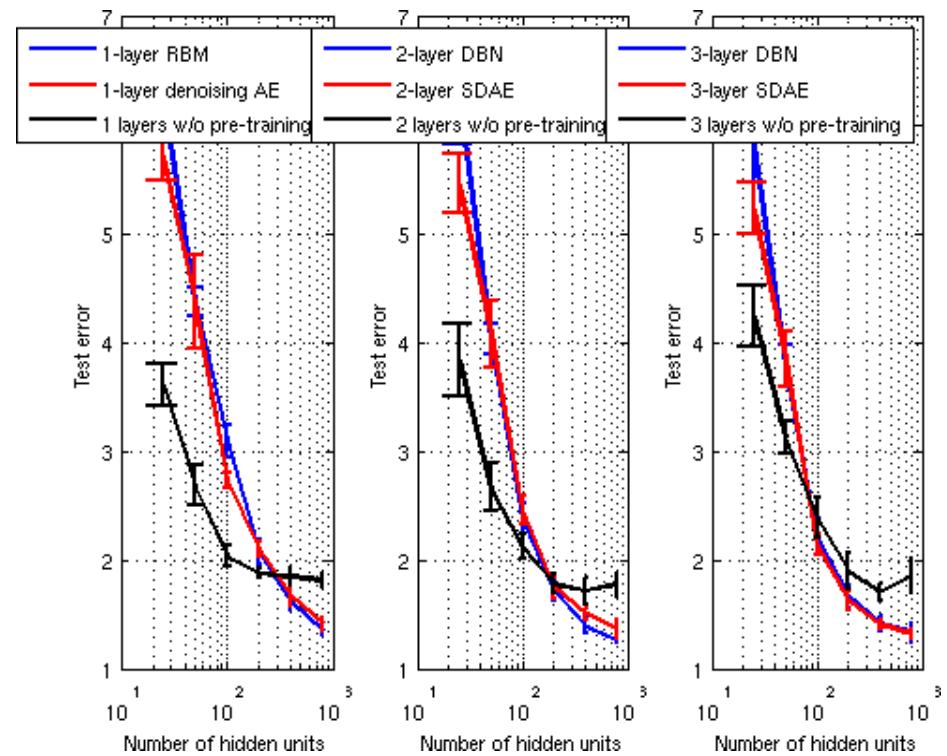
Learning Trajectories in Function Space

- 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



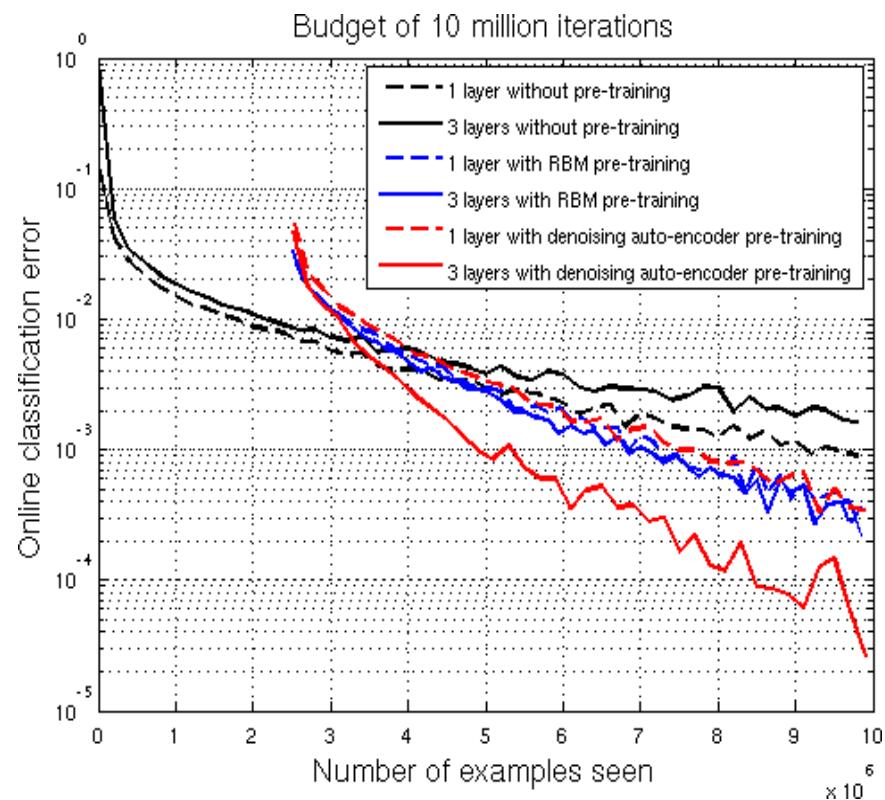
Unsupervised learning as regularizer

- 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

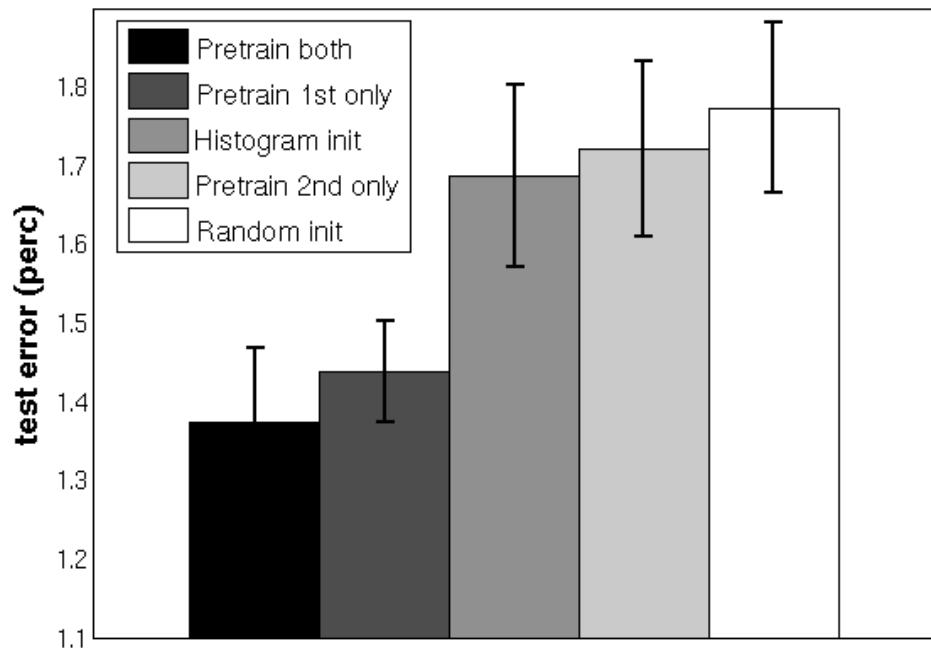


Better optimization of online error

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



Pre-training lower layers more critical



Verifies that what matters is not just the marginal distribution over initial weight values
(Histogram init.)

The Credit Assignment Problem

- Even with the correct gradient, lower layers (far from the prediction, close to input) are the most difficult to train
- Lower layers benefit most from unsupervised pre-training
 - Local unsupervised signal = extract / disentangle factors
 - Temporal constancy
 - Mutual information between multiple modalities
- Credit assignment / error information not flowing easily?
- Related to difficulty of credit assignment through time?

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

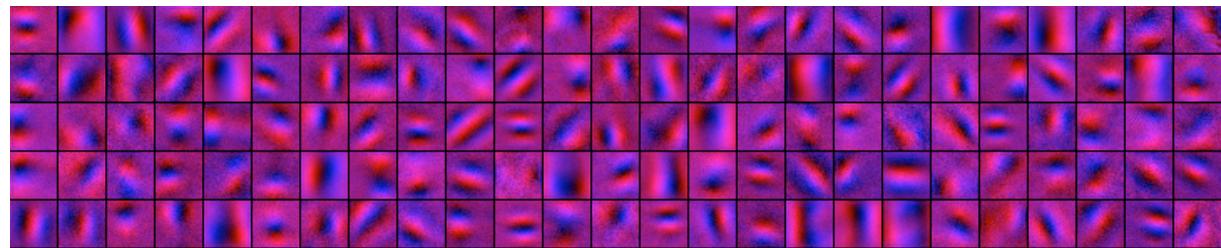
Semi-Supervised Embedding

- Use pairs (or triplets) of examples which are known to represent nearby concepts (or not)
- Bring closer the intermediate representations of supposedly similar pairs, push away the representations of randomly chosen pairs
- (Weston, Ratle & Collobert, ICML'2008): improved semi-supervised learning by combining unsupervised embedding criterion with supervised gradient

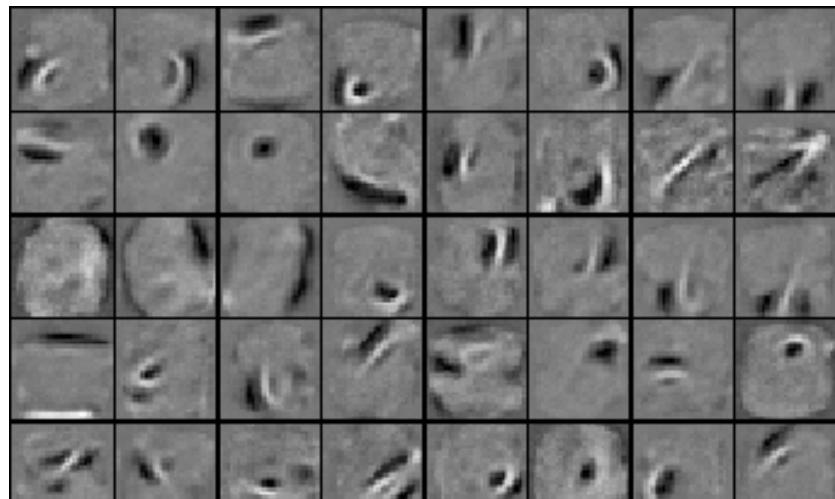
Slow Features

- Successive images in a video = similar
- Randomly chosen pair of images = dissimilar
- Slowly varying features are likely to represent interesting abstractions

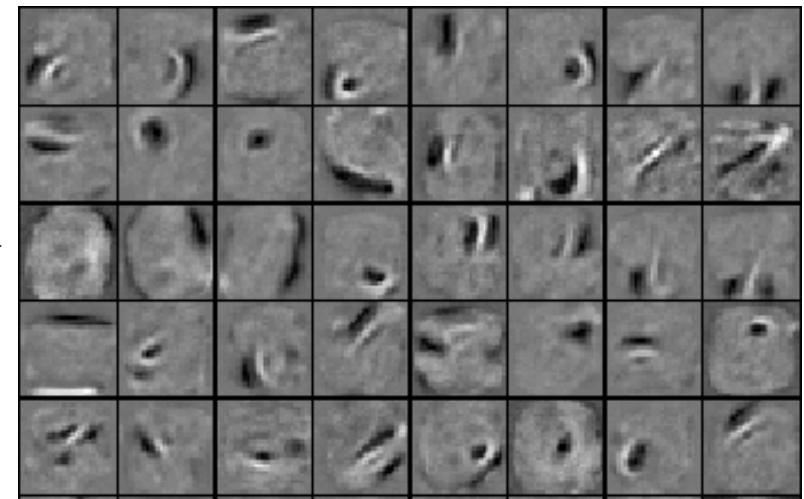
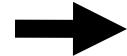
Slow features
1st layer



Learning Dynamics of Deep Nets



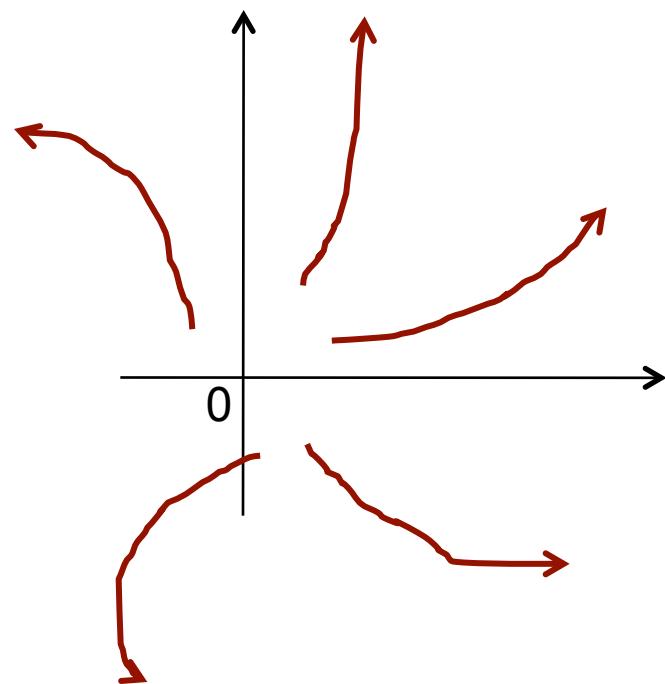
Before fine-tuning



After fine-tuning

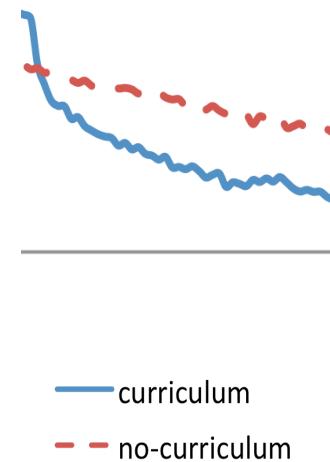
Learning Dynamics of Deep Nets

- 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

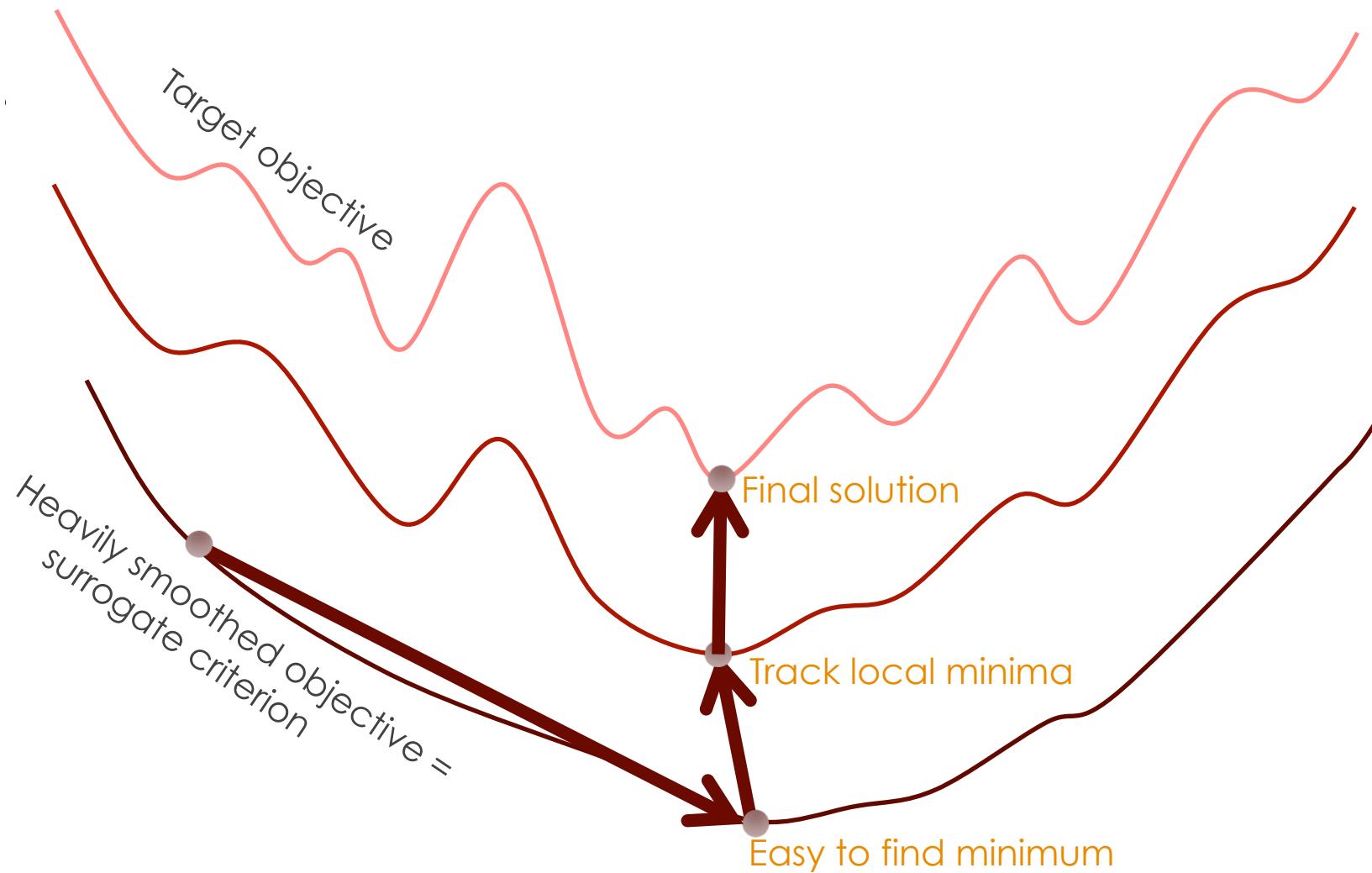


Order & Selection of Examples Matters

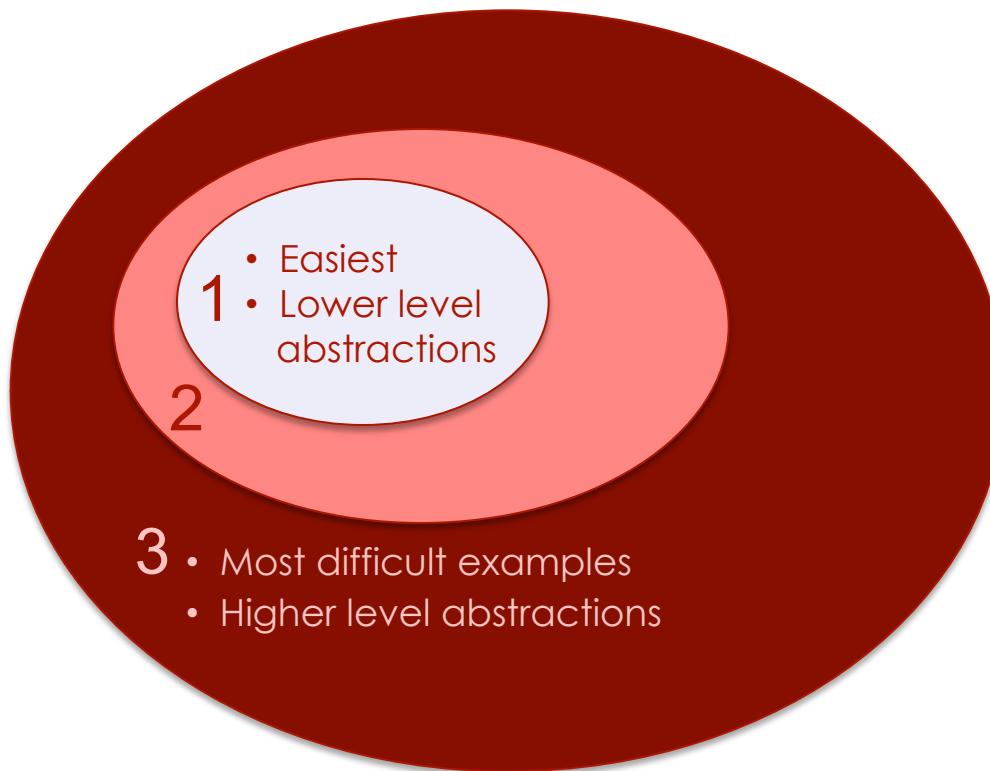
- Curriculum learning
(Bengio et al, ICML'2009; Krueger & Dayan 2009)
- Start with easier examples
- Faster convergence to a better local minimum in deep architectures
- Also acts like a regularizer with optimization effect?
- Influencing learning dynamics can make a big difference



Continuation Methods



Curriculum Learning as Continuation



- Sequence of training distributions
- Initially peaking on easier / simpler ones
- Gradually give more weight to more difficult ones until reach target distribution

Take-Home Messages

- Break-through in learning complicated functions:
deep architectures with distributed representations
- Multiple levels of latent variables:
potentially exponential gain in statistical sharing
- Main challenge: training deep architectures
- RBMs allow fast inference, stacked RBMs / auto-encoders
have fast approximate inference
- 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?
- Capturing scene structure and sequential structure?

Thank you for your attention!

- Questions?
- Comments?