

Representation Learning

Yoshua Bengio

ICML 2012 Tutorial

June 26th 2012, Edinburgh, Scotland



Outline of the Tutorial

1. Motivations and Scope

1. Feature / Representation learning
2. Distributed representations
3. Exploiting unlabeled data
4. Deep representations
5. Multi-task / Transfer learning
6. Invariance vs Disentangling

2. Algorithms

1. Probabilistic models and RBM variants
2. Auto-encoder variants (sparse, denoising, contractive)
3. Explaining away, sparse coding and Predictive Sparse Decomposition
4. Deep variants

3. Analysis, Issues and Practice

1. Tips, tricks and hyper-parameters
2. Partition function gradient
3. Inference
4. Mixing between modes
5. Geometry and probabilistic Interpretations of auto-encoders
6. Open questions

See (Bengio, Courville & Vincent 2012)

“Unsupervised Feature Learning and Deep Learning: A Review and New Perspectives”

And <http://www.iro.umontreal.ca/~bengioy/talks/deep-learning-tutorial-2012.html> for a detailed list of references.

Ultimate Goals

- AI
- Needs knowledge
- Needs learning
- Needs generalizing where probability mass concentrates
- Needs ways to fight the curse of dimensionality
- Needs disentangling the underlying explanatory factors (“making sense of the data”)

Representing data

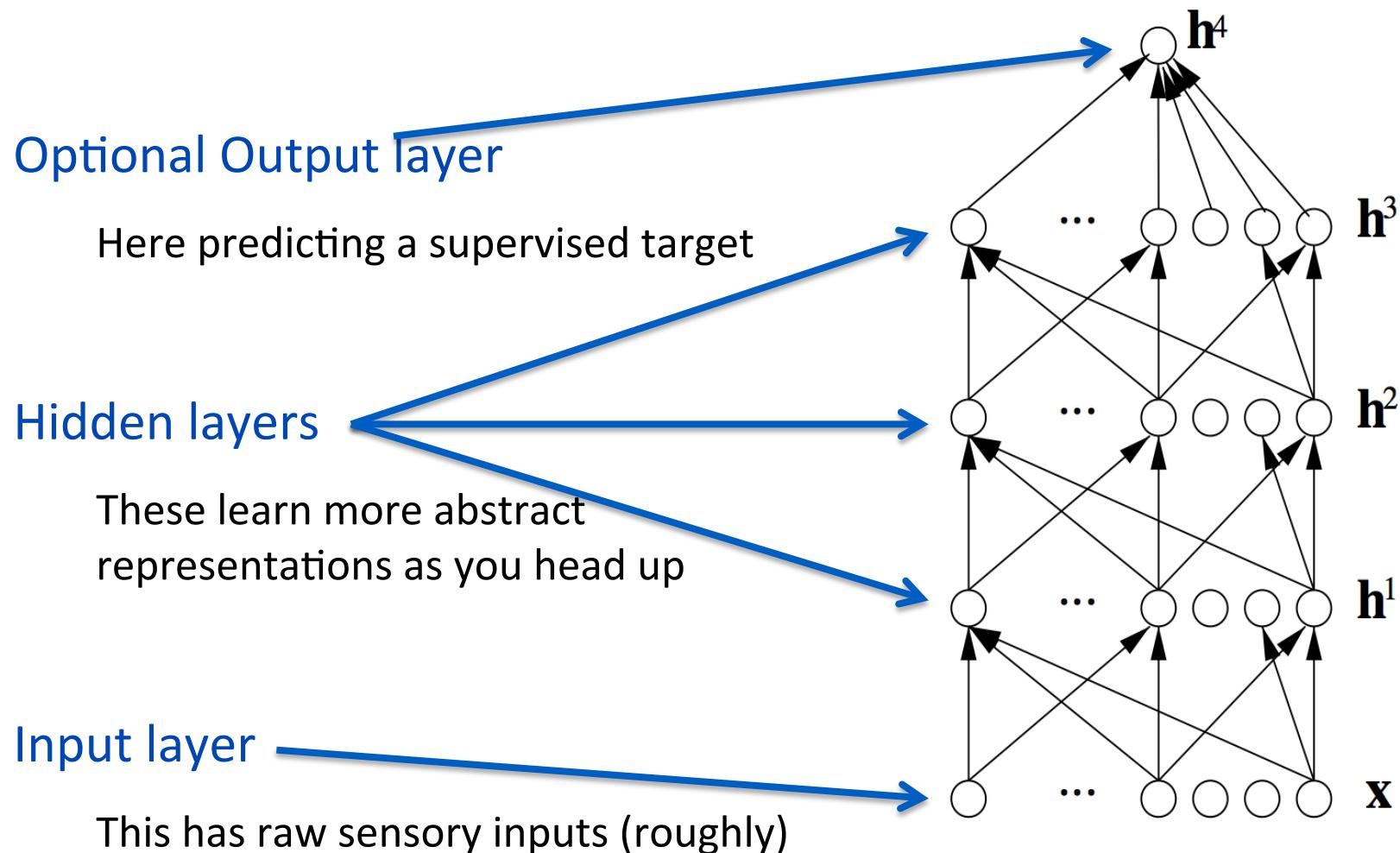
- In practice ML very sensitive to choice of data representation
 - feature engineering (where most effort is spent)
 - (better) feature learning (this talk):
automatically learn good representations
- Probabilistic models:
 - Good representation = captures *posterior distribution of underlying explanatory factors of observed input*
- Good features are useful to explain variations

Deep Representation Learning

Deep learning algorithms attempt to learn multiple levels of representation of increasing complexity/abstraction

When the number of levels can be data-selected, this is a deep architecture

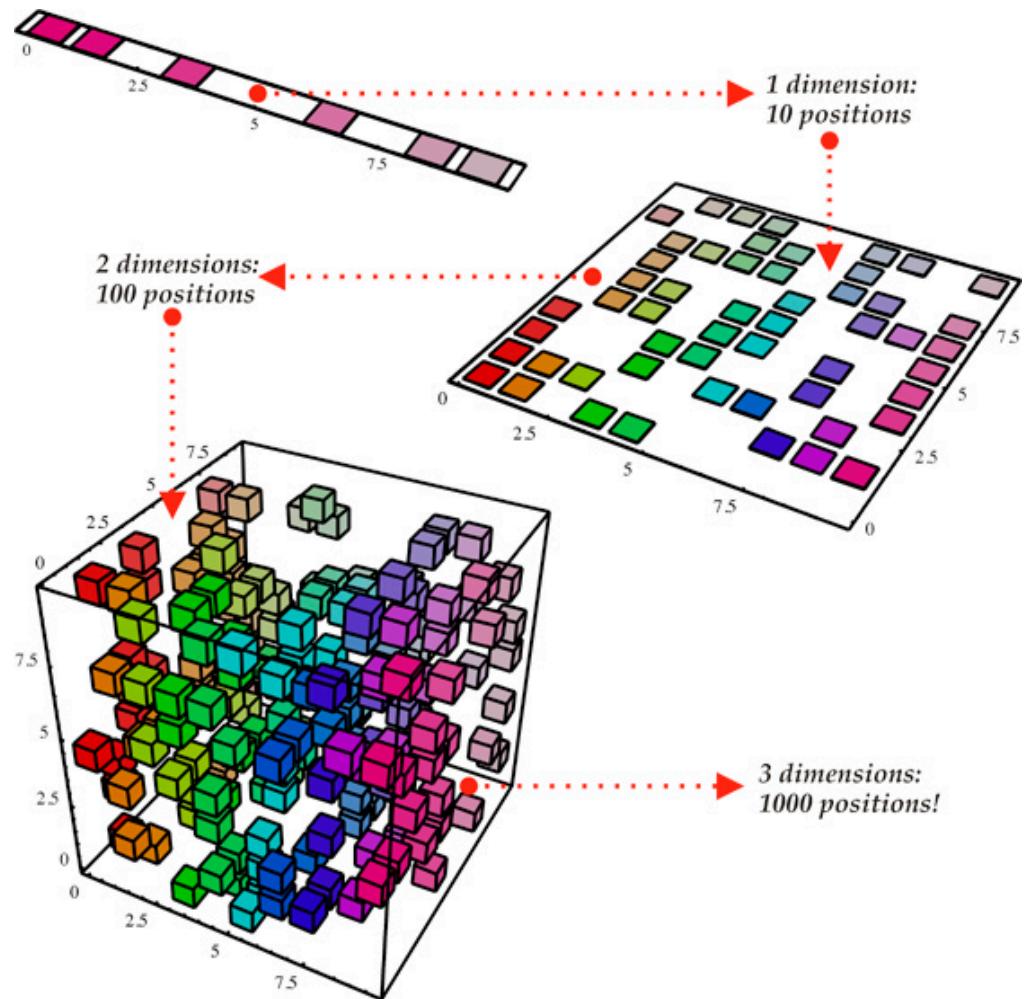
A Good Old Deep Architecture



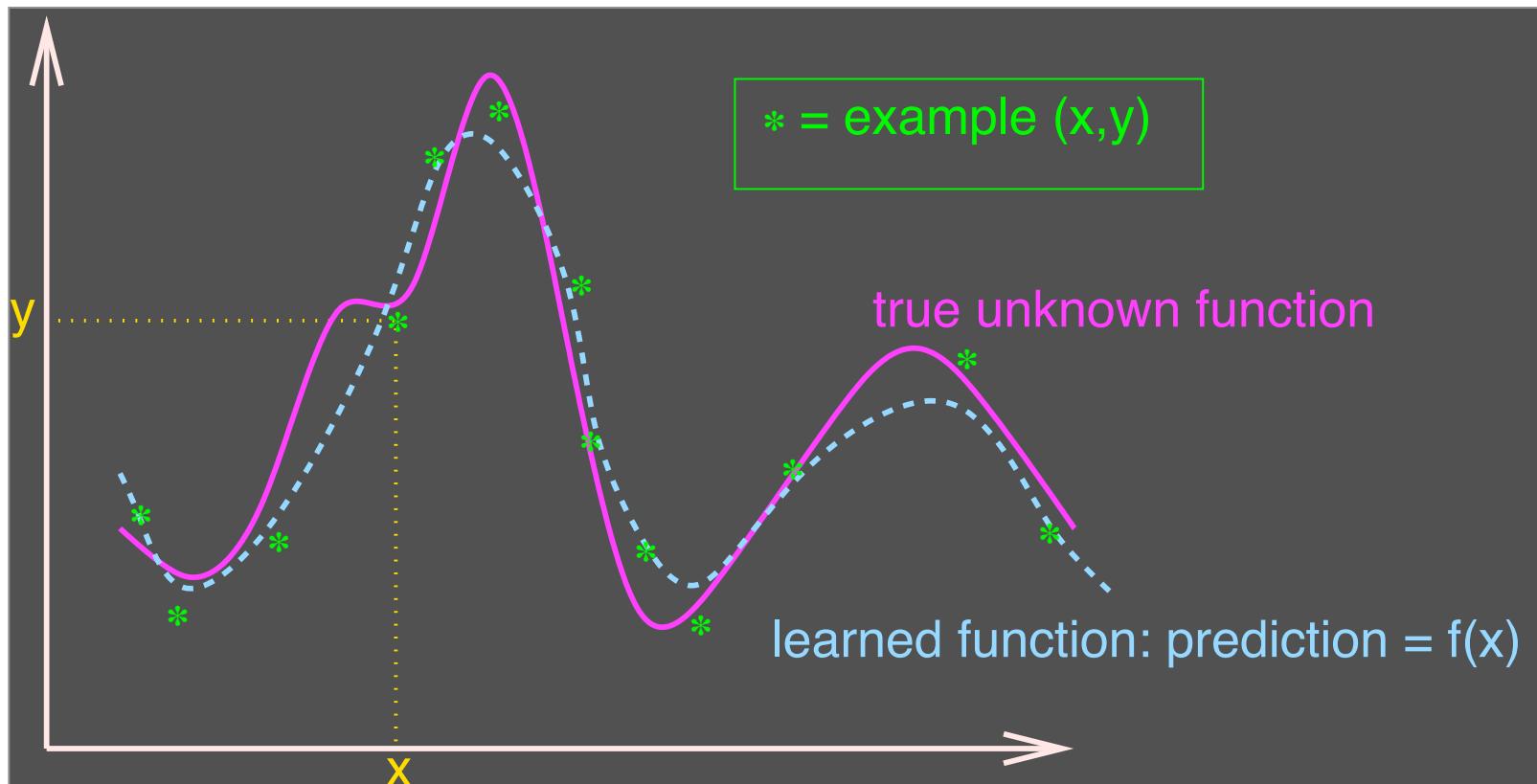
What We Are Fighting Against: The Curse of Dimensionality

To generalize locally,
need representative
examples for all
relevant variations!

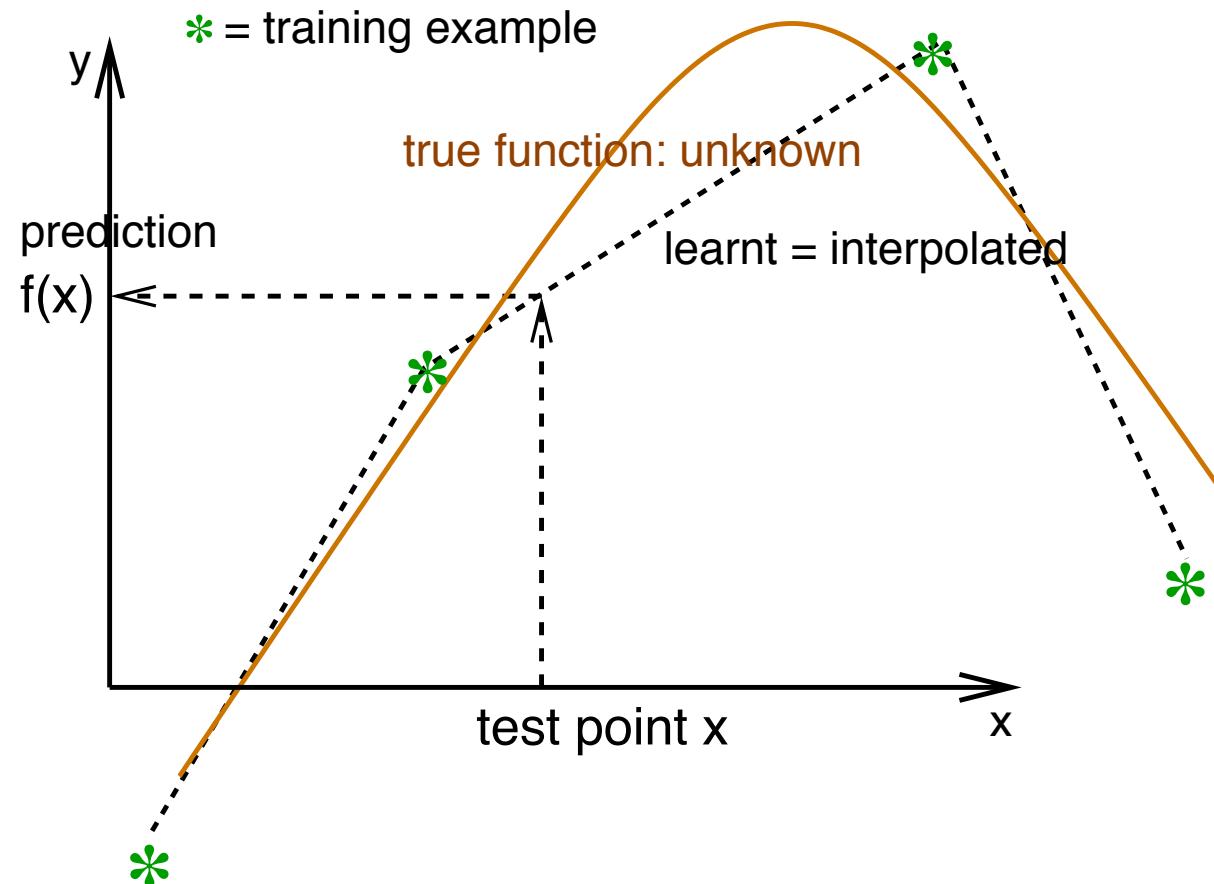
Classical solution: hope
for a smooth enough
target function, or
make it smooth by
handcrafting features



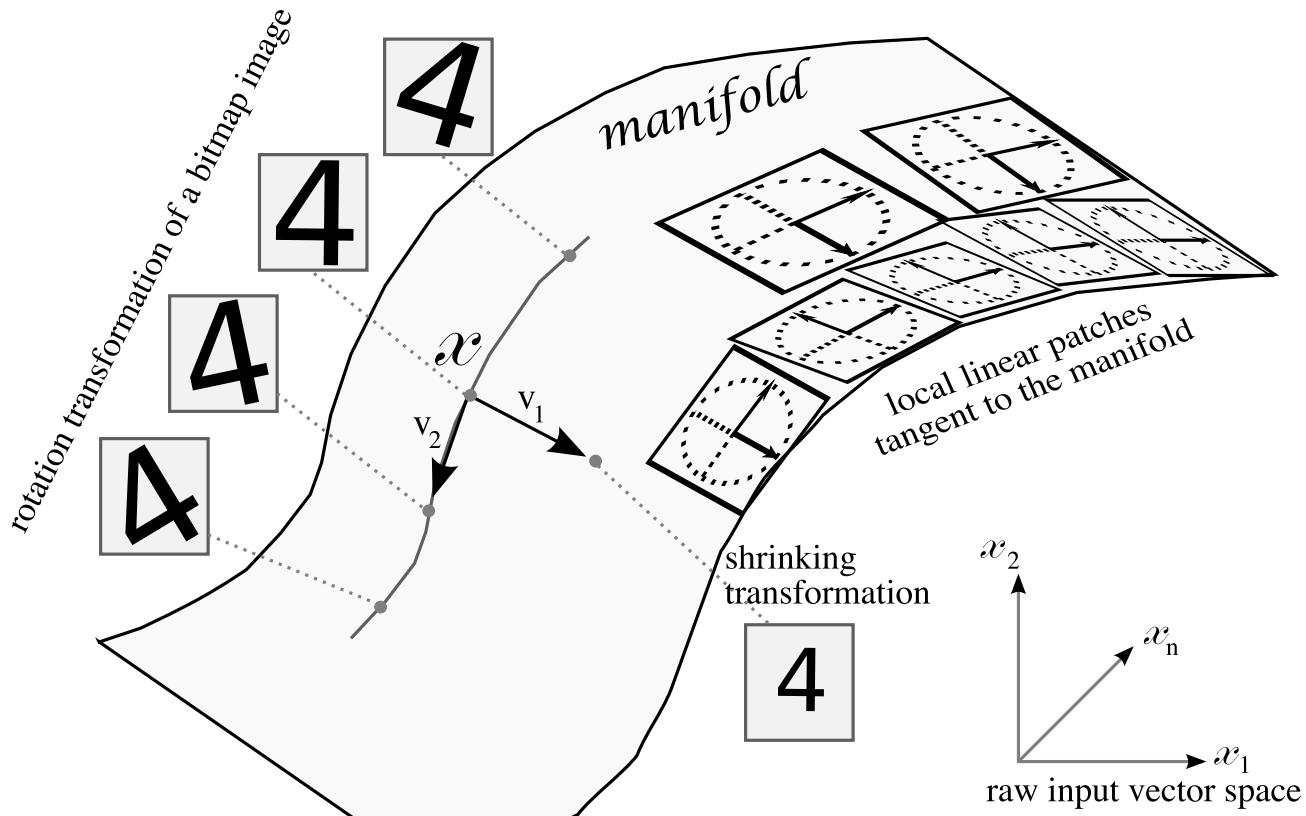
Easy Learning



Local Smoothness Prior: Locally Capture the Variations



Real Data Are on Highly Curved Manifolds

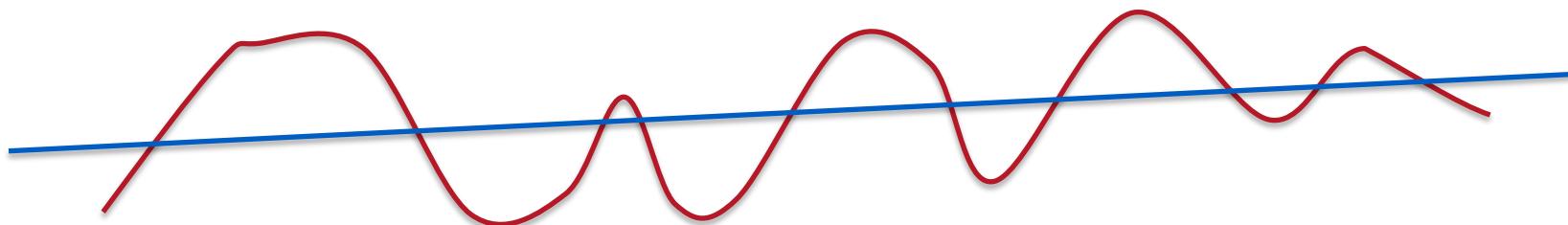


Not Dimensionality so much as Number of Variations



(Bengio, Delalleau & Le Roux 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 requires $O(2^d)$ examples

Is there any hope to
generalize non-locally?

Yes! Need more priors!

Part 1

Six Good Reasons to Explore Representation Learning

#1 Learning features, not just handcrafting them

Most ML systems use very carefully hand-designed features and representations

Many practitioners are very experienced – and good – at such feature design (or kernel design)

In this world, “machine learning” reduces mostly to linear models (including CRFs) and nearest-neighbor-like features/models (including n-grams, kernel SVMs, etc.)

Hand-crafting features is time-consuming, brittle, incomplete

How can we automatically learn good features?

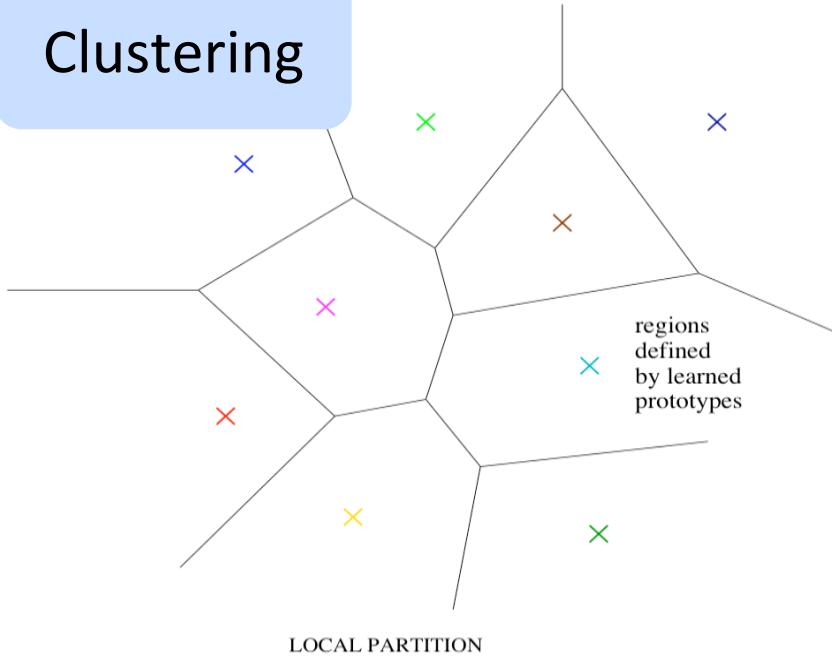
Claim: to approach AI, need to move scope of ML beyond hand-crafted features and simple models

Humans develop representations and abstractions to enable problem-solving and reasoning; our computers should do the same

Handcrafted features can be combined with learned features, or new more abstract features learned on top of handcrafted features

#2 The need for distributed representations

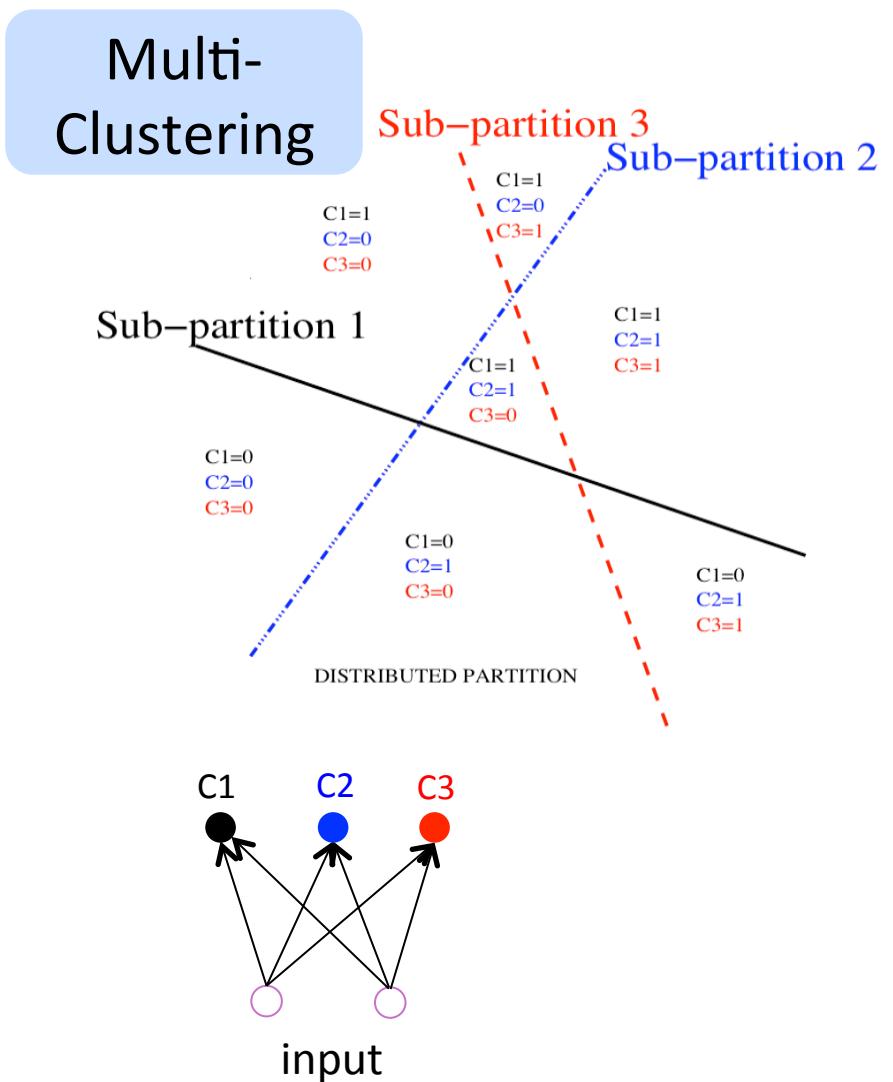
Clustering



- Clustering, Nearest-Neighbors, RBF SVMs, local non-parametric density estimation & prediction, decision trees, etc.
- Parameters for each distinguishable region
- # distinguishable regions linear in # parameters

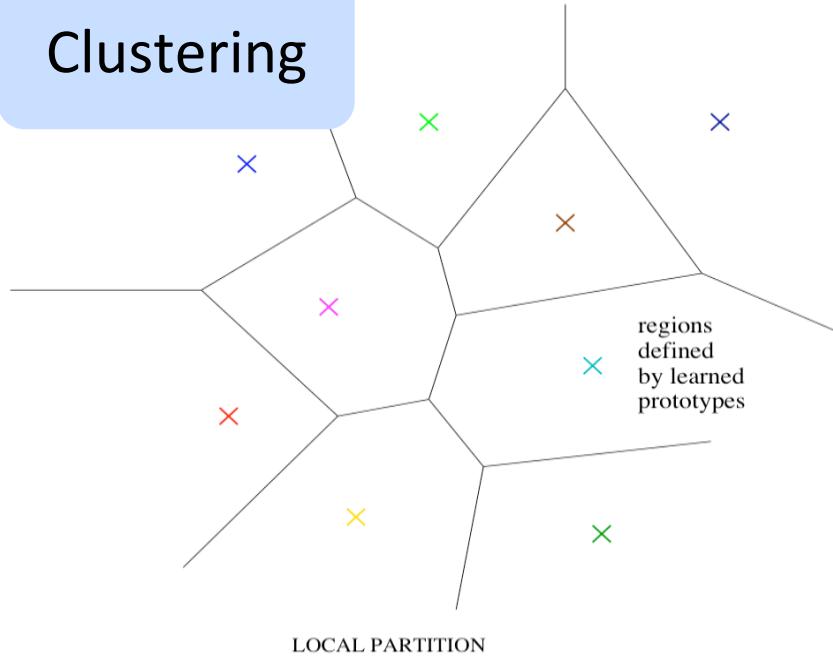
#2 The need for distributed representations

- Factor models, PCA, RBMs, Neural Nets, Sparse Coding, Deep Learning, etc.
- Each parameter influences many regions, not just local neighbors
- # distinguishable regions grows almost exponentially with # parameters
- **GENERALIZE NON-LOCALLY TO NEVER-SEEN REGIONS**

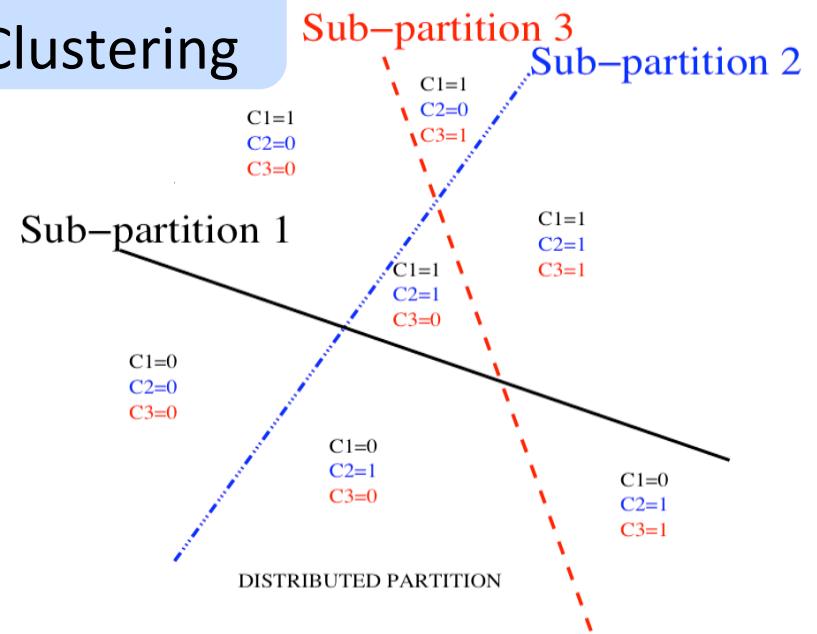


#2 The need for distributed representations

Clustering



Multi-Clustering



Learning a **set of features** that are not mutually exclusive can be **exponentially more statistically efficient** than nearest-neighbor-like or clustering-like models

#3 Unsupervised feature Learning

Today, most practical ML applications require (lots of) labeled training data

But almost all **data is unlabeled**

The brain needs to learn about 10^{14} synaptic strengths

... in about 10^9 seconds

Labels cannot possibly provide enough information

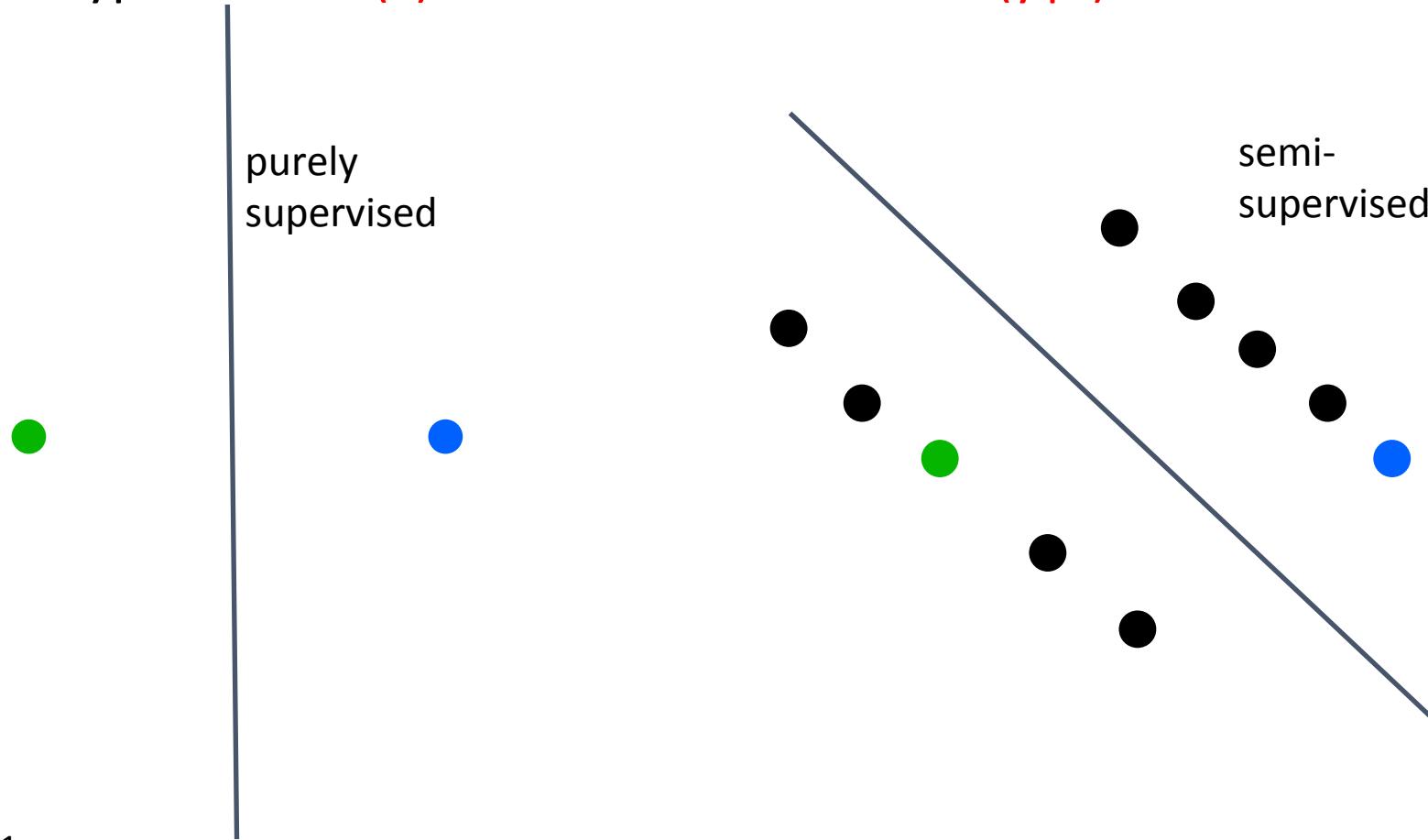
Most information acquired in an **unsupervised** fashion

#3 How do humans generalize from very few examples?

- They **transfer** knowledge from previous learning:
 - Representations
 - Explanatory factors
- Previous learning from: unlabeled data
 - + labels for other tasks
- **Prior: shared underlying explanatory factors, in particular between $P(x)$ and $P(Y|x)$**

#3 Sharing Statistical Strength by Semi-Supervised Learning

- Hypothesis: $P(x)$ shares structure with $P(y|x)$



#4 Learning multiple levels of representation

There is theoretical and empirical evidence in favor of multiple levels of representation

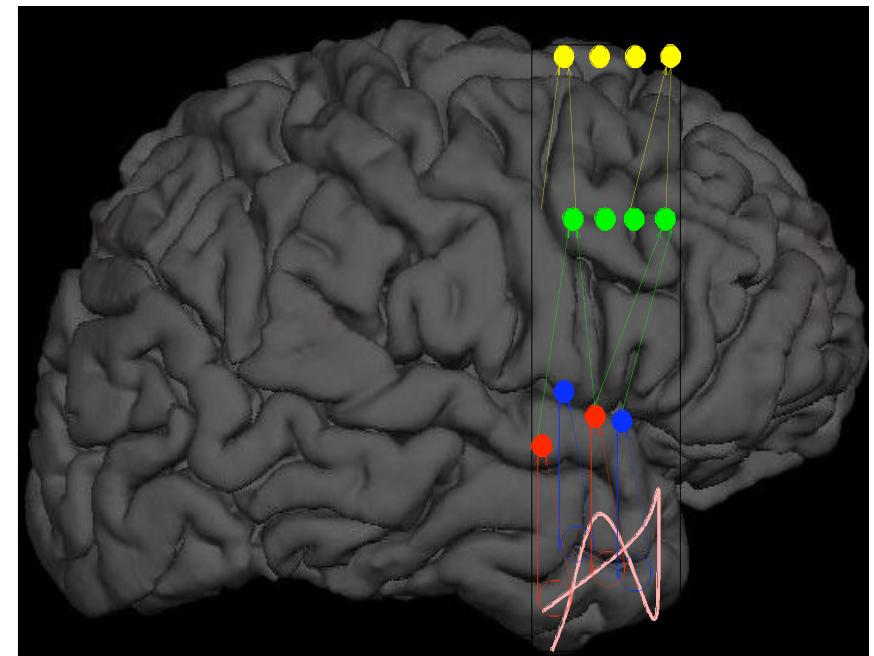
Exponential gain for some families of functions

Biologically inspired learning

Brain has a deep architecture

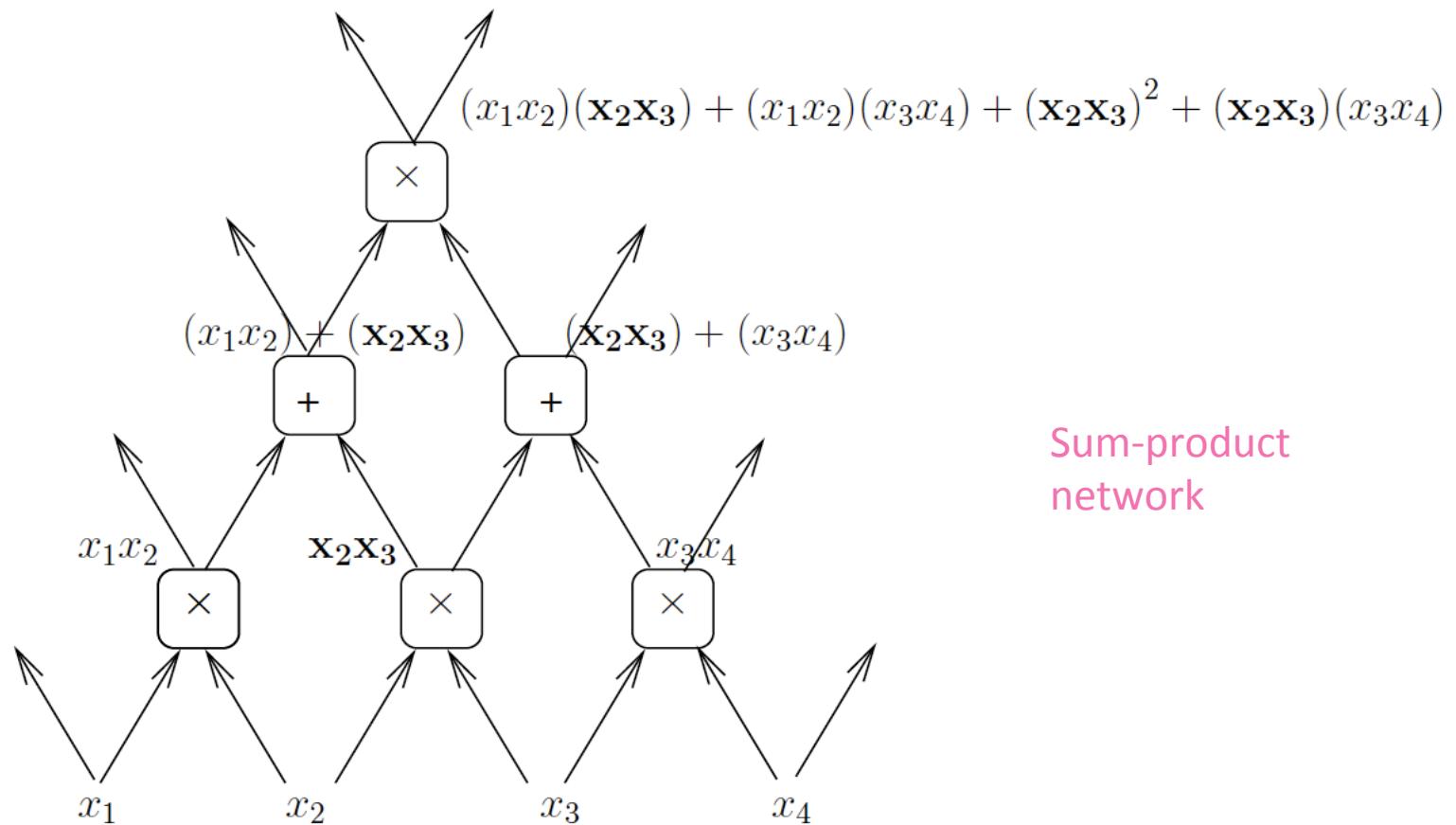
Cortex seems to have a generic learning algorithm

Humans first learn simpler concepts and then compose them to more complex ones



#4 Sharing Components in a Deep Architecture

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

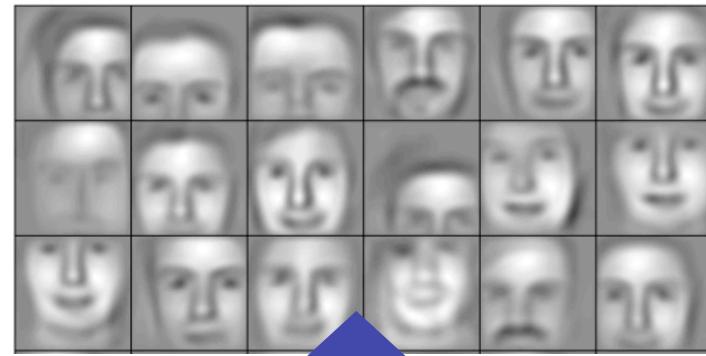


#4 Learning multiple levels of representation



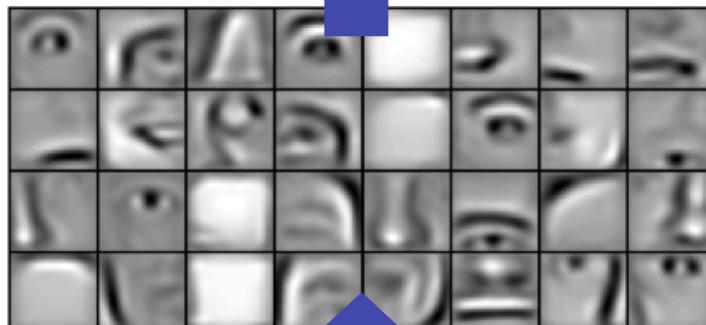
(Lee, Largman, Pham & Ng, NIPS 2009)
(Lee, Grosse, Ranganath & Ng, ICML 2009)

Successive model layers learn deeper intermediate representations



Layer 3

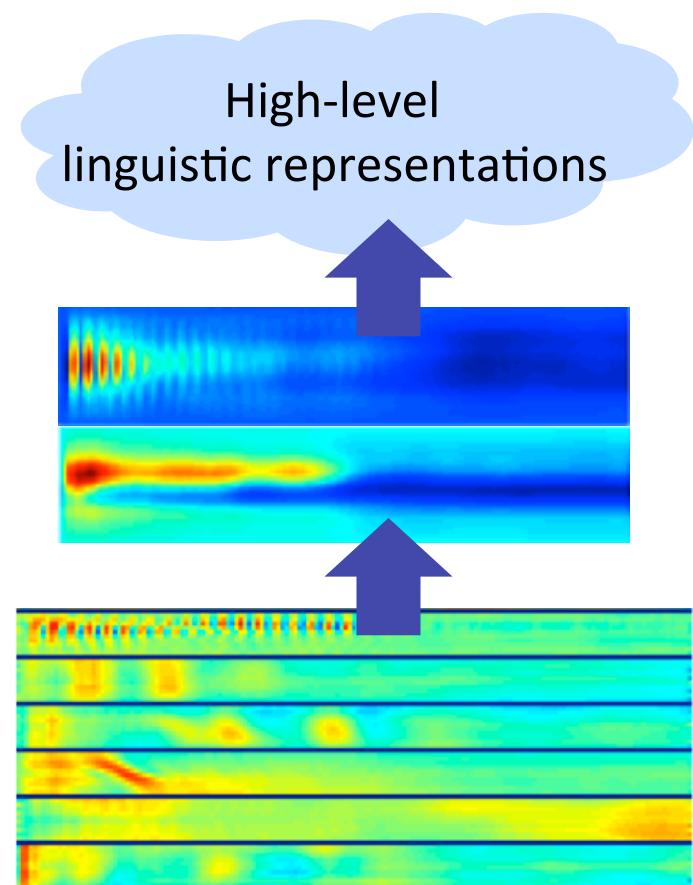
Parts combine
to form objects



Layer 2

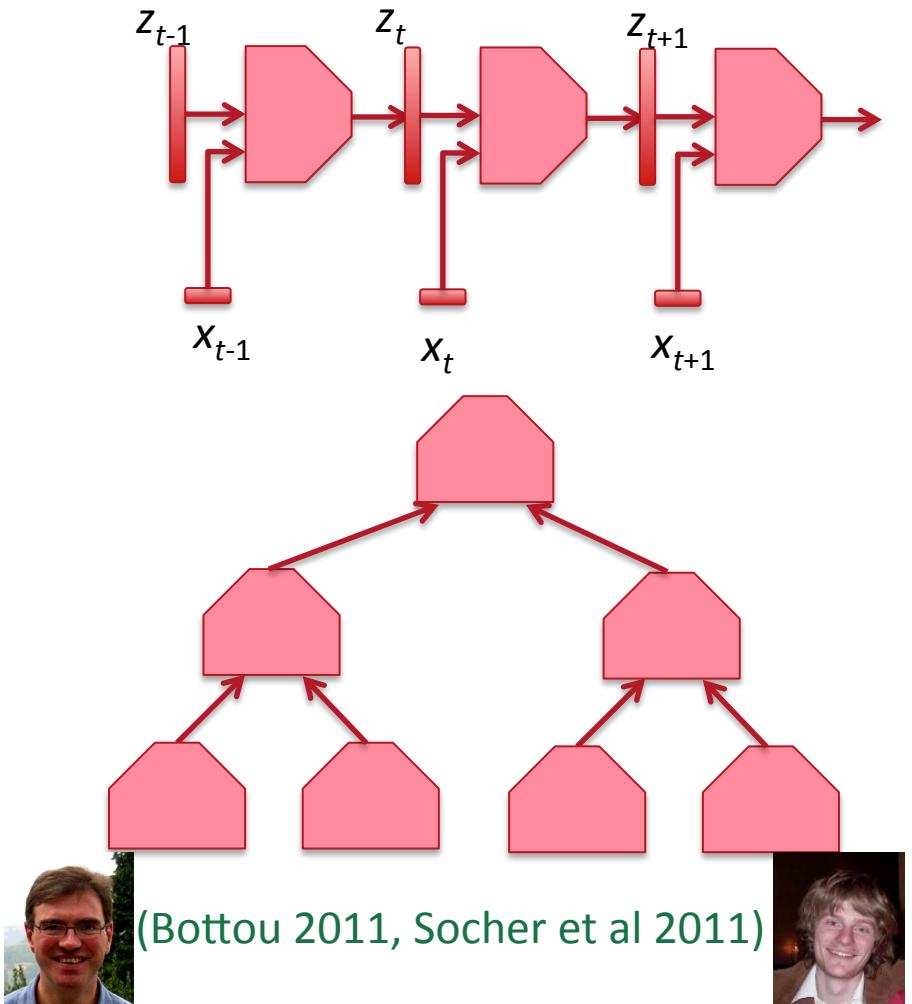


Layer 1



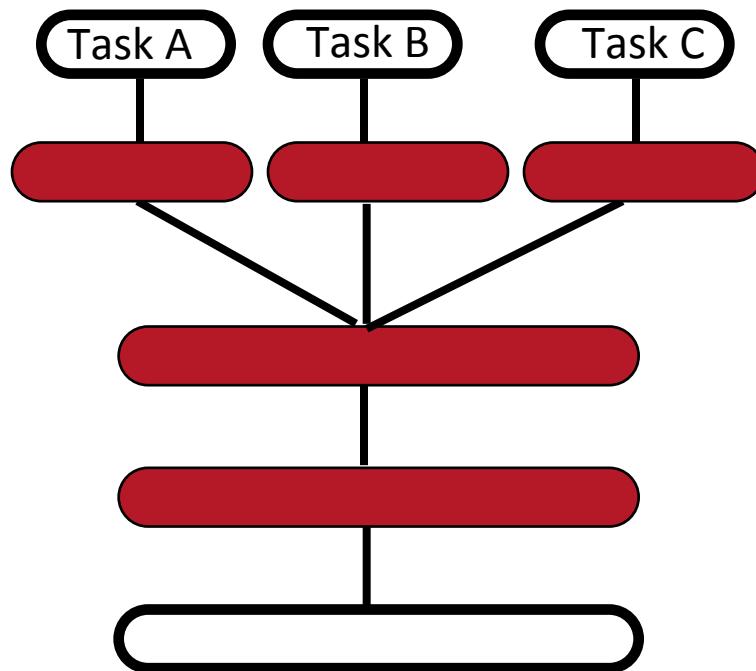
#4 Handling the compositionality of human language and thought

- Human languages, ideas, and artifacts are composed from simpler components
- Recursion: the same operator (same parameters) is applied repeatedly on different states/components of the computation
- Result after unfolding = deep representations



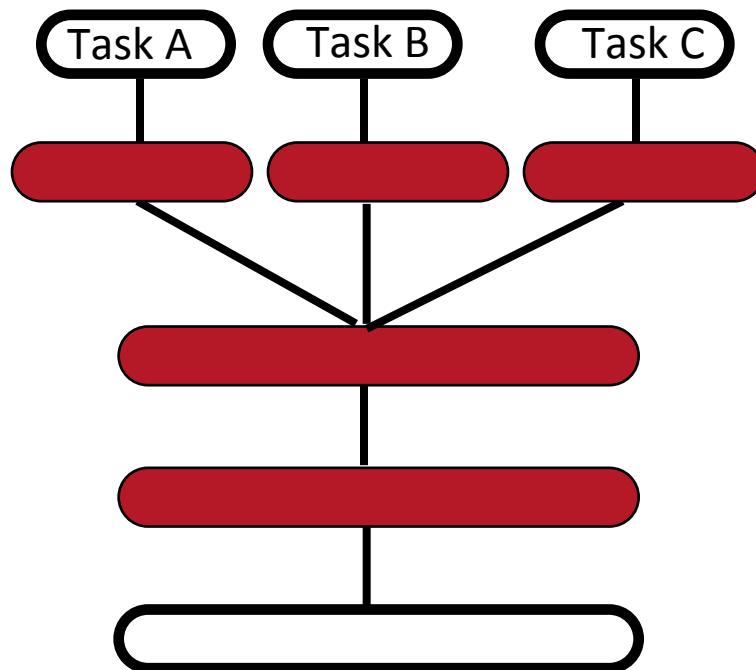
#5 Multi-Task Learning

- Generalizing better to new tasks is crucial to approach AI
- Deep architectures learn good intermediate representations that can be shared across tasks
- Good representations that disentangle underlying factors of variation make sense for many tasks because each task concerns a subset of the factors



#5 Sharing Statistical Strength

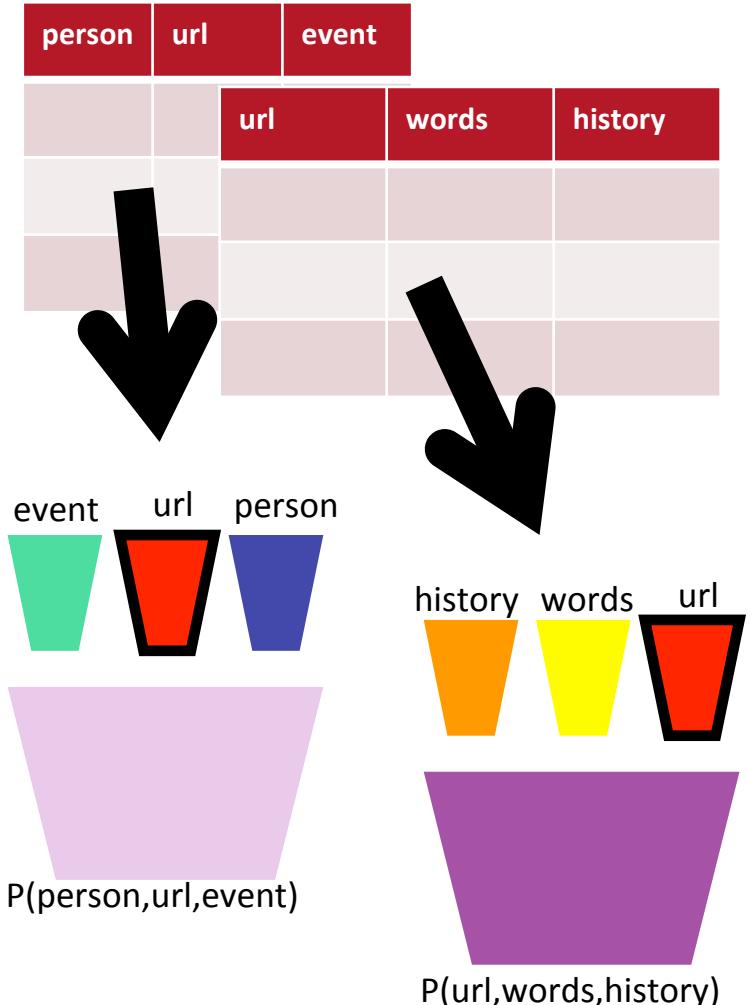
- Multiple levels of latent variables also allow combinatorial sharing of statistical strength: intermediate levels can also be seen as sub-tasks
- E.g. dictionary, with intermediate concepts re-used across many definitions



Prior: some shared underlying explanatory factors between tasks

#5 Combining Multiple Sources of Evidence with Shared Representations

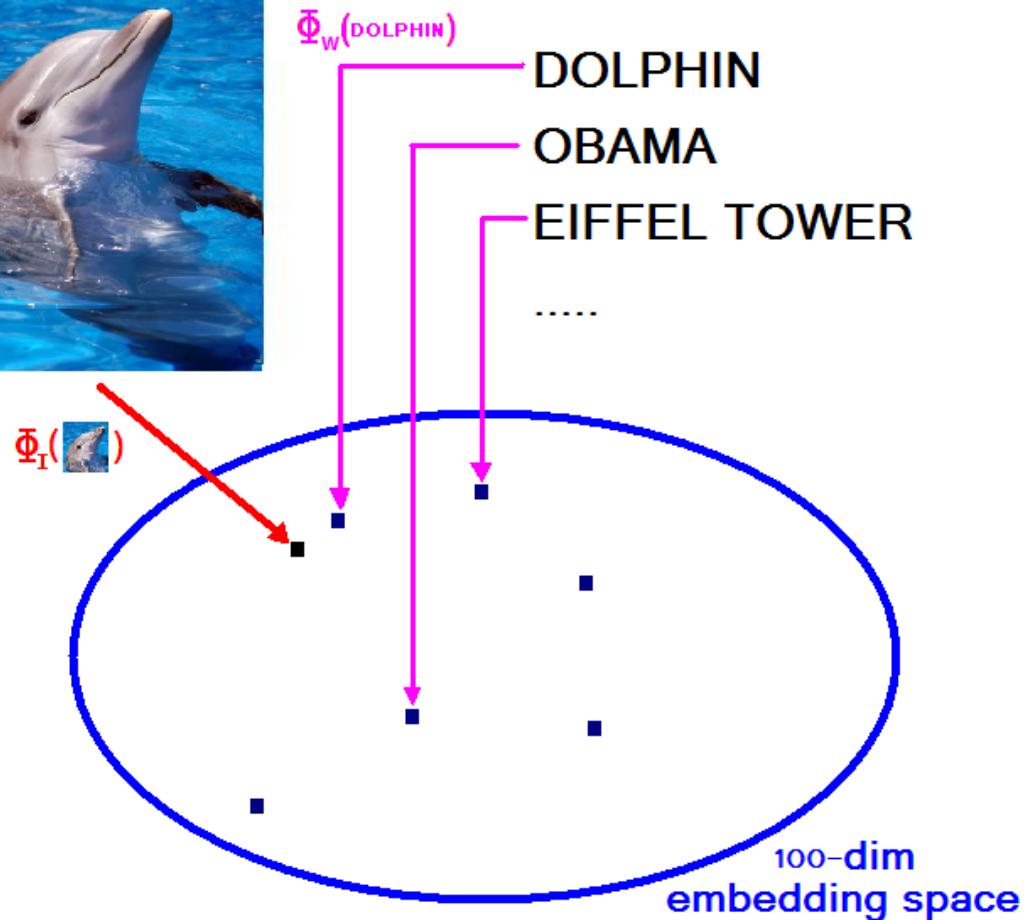
- Traditional ML: data = matrix
- Relational learning: multiple sources, different tuples of variables
- Share representations of same types across data sources
- Shared learned representations help propagate information among data sources: e.g., WordNet, XWN, Wikipedia, FreeBase, ImageNet...
(Bordes et al AISTATS 2012)



#5 Different object types represented in same space



Google:
S. Bengio, J.
Weston & N.
Usunier
(IJCAI 2011,
NIPS'2010,
JMLR 2010,
MLJ 2010)



Learn $\Phi_T(\cdot)$ and $\Phi_w(\cdot)$ to optimize precision@k.

#6 Invariance and Disentangling

- Invariant features
- Which invariances?
- Alternative: learning to disentangle factors
- Good disentangling →
 avoid the curse of dimensionality



#6 Emergence of Disentangling

- (Goodfellow et al. 2009): sparse auto-encoders trained on images
 - some higher-level features more invariant to geometric factors of variation
- (Glorot et al. 2011): sparse rectified denoising auto-encoders trained on bags of words for sentiment analysis
 - different features specialize on different aspects (domain, sentiment)



WHY?

#6 Sparse Representations

- Just add a penalty on learned representation
- Information disentangling (compare to dense compression)
- More likely to be linearly separable (high-dimensional space)
- Locally low-dimensional representation = local chart
- Hi-dim. sparse = efficient **variable size** representation
= **data structure**

Few bits of information



Many bits of information



Prior: only few concepts and attributes relevant per example

Bypassing the curse

We need to build **compositionality** into our ML models

Just as human languages exploit compositionality to give representations and meanings to complex ideas

Exploiting compositionality gives an exponential gain in representational power

Distributed representations / embeddings: **feature learning**

Deep architecture: **multiple levels of feature learning**

Prior: compositionality is useful to describe the world around us efficiently

Bypassing the curse by sharing statistical strength

- Besides very fast GPU-enabled predictors, the main advantage of representation learning is **statistical**: potential to learn from less labeled examples because of sharing of statistical strength:
 - Unsupervised pre-training and semi-supervised training
 - Multi-task learning
 - Multi-data sharing, learning about symbolic objects and their relations

Why now?

Despite prior investigation and understanding of many of the algorithmic techniques ...

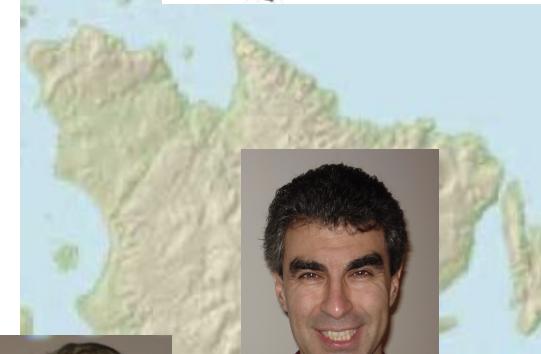
Before 2006 training deep architectures was **unsuccessful**
(except for convolutional neural nets when used by people who speak French)

What has changed?

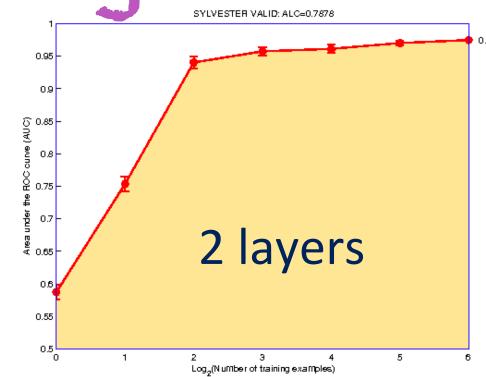
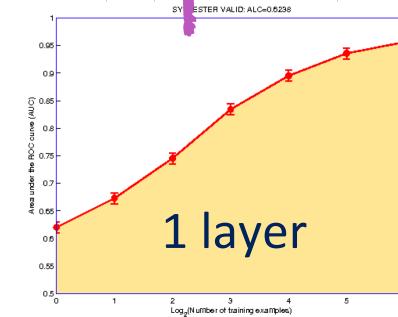
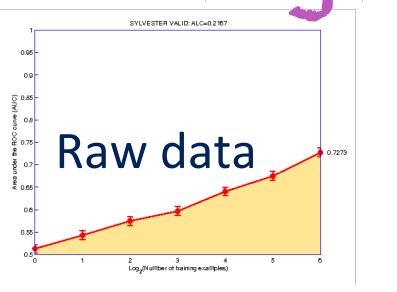
- New methods for unsupervised pre-training have been developed (variants of Restricted Boltzmann Machines = RBMs, regularized autoencoders, sparse coding, etc.)
- Better understanding of these methods
- Successful real-world applications, winning challenges and beating SOTAs in various areas

Major Breakthrough in 2006

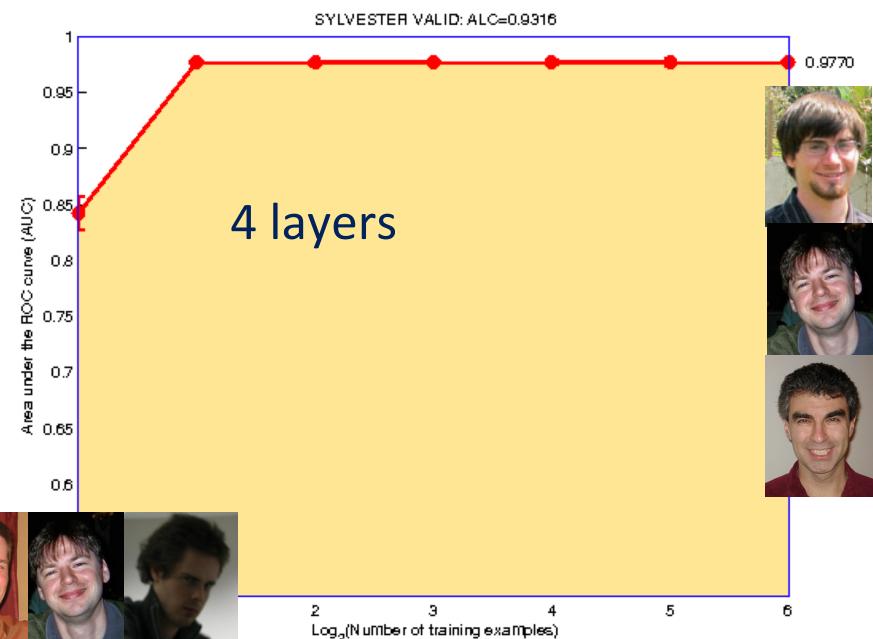
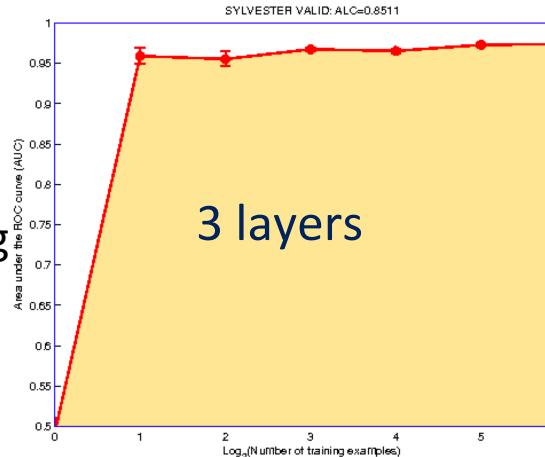
- Ability to train deep architectures by using layer-wise unsupervised learning, whereas previous purely supervised attempts had failed
- Unsupervised feature learners:
 - RBMs
 - Auto-encoder variants
 - Sparse coding variants



Unsupervised and Transfer Learning Challenge + Transfer Learning Challenge: Deep Learning 1st Place

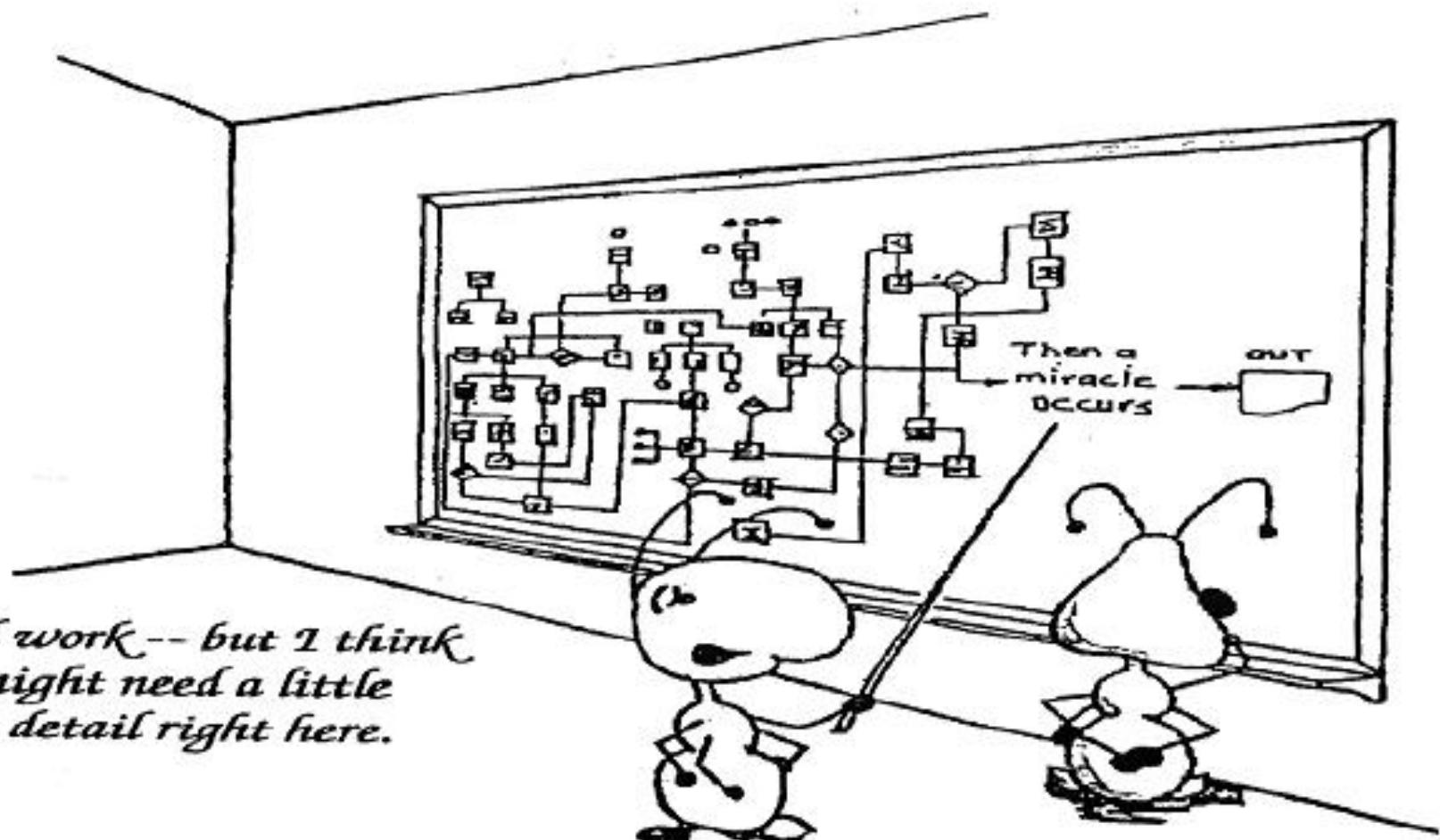


ICML'2011 workshop on Unsup. & Transfer Learning



More Successful Applications

- Microsoft uses DL for speech rec. service (audio video indexing), based on Hinton/Toronto's DBNs ([Mohamed et al 2011](#))
- Google uses DL in its Google Goggles service, using Ng/Stanford DL systems
- NYT today talks about these: http://www.nytimes.com/2012/06/26/technology/in-a-big-network-of-computers-evidence-of-machine-learning.html?_r=1
- Substantially beating SOTA in language modeling (perplexity from 140 to 102 on Broadcast News) for speech recognition (WSJ WER from 16.9% to 14.4%) ([Mikolov et al 2011](#)) and translation (+1.8 BLEU) ([Schwenk 2012](#))
- SENNA: Unsup. pre-training + multi-task DL reaches SOTA on POS, NER, SRL, chunking, parsing, with >10x better speed & memory ([Collobert et al 2011](#))
- Recursive nets surpass SOTA in paraphrasing ([Socher et al 2011](#))
- Denoising AEs substantially beat SOTA in sentiment analysis ([Glorot et al 2011](#))
- Contractive AEs SOTA in knowledge-free MNIST (.8% err) ([Rifai et al NIPS 2011](#))
- Le Cun/NYU's stacked PSDs most accurate & fastest in pedestrian detection and DL in top 2 winning entries of German road sign recognition competition

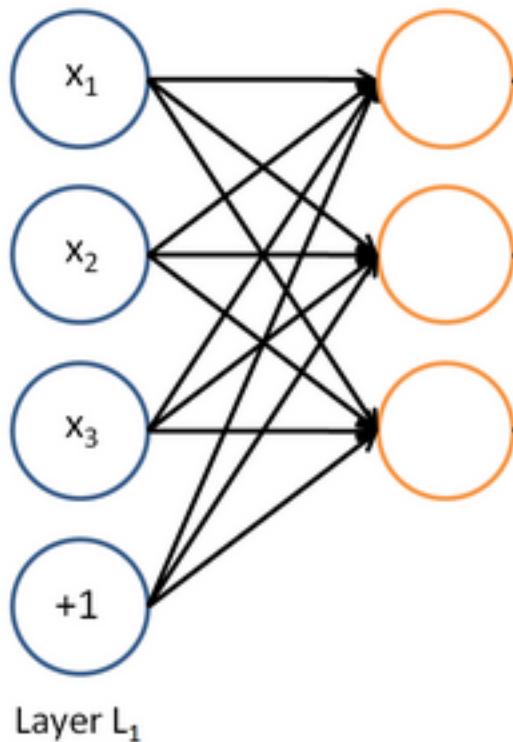


Part 2

Representation Learning Algorithms

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

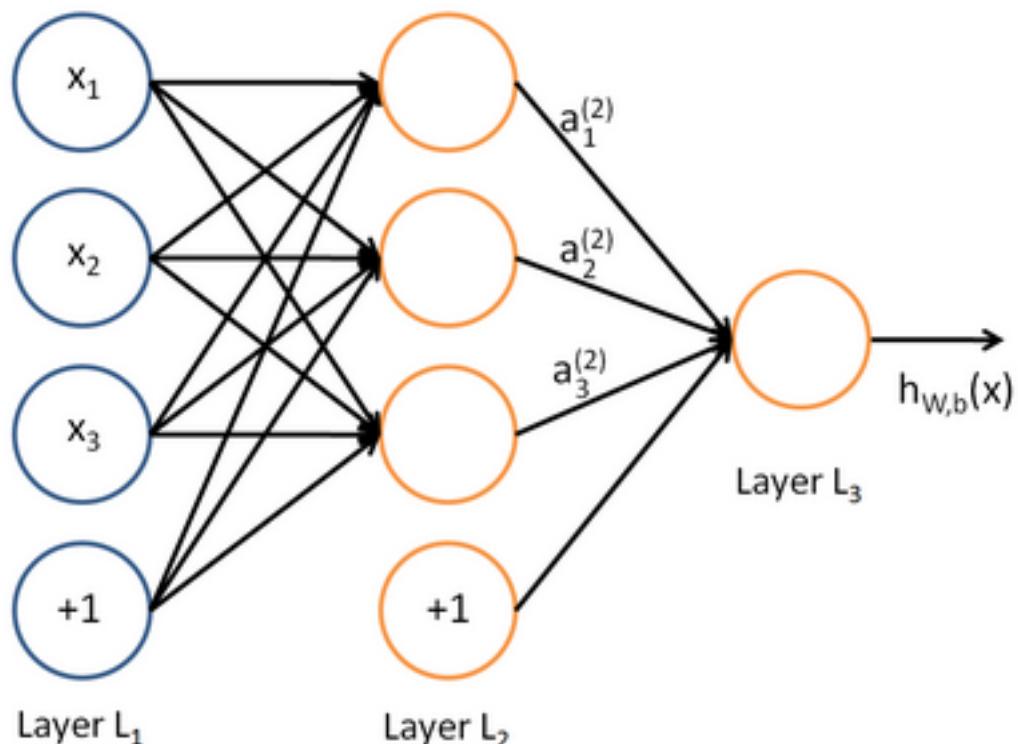
If we feed a vector of inputs through a bunch of logistic regression functions, then we get a vector of outputs



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

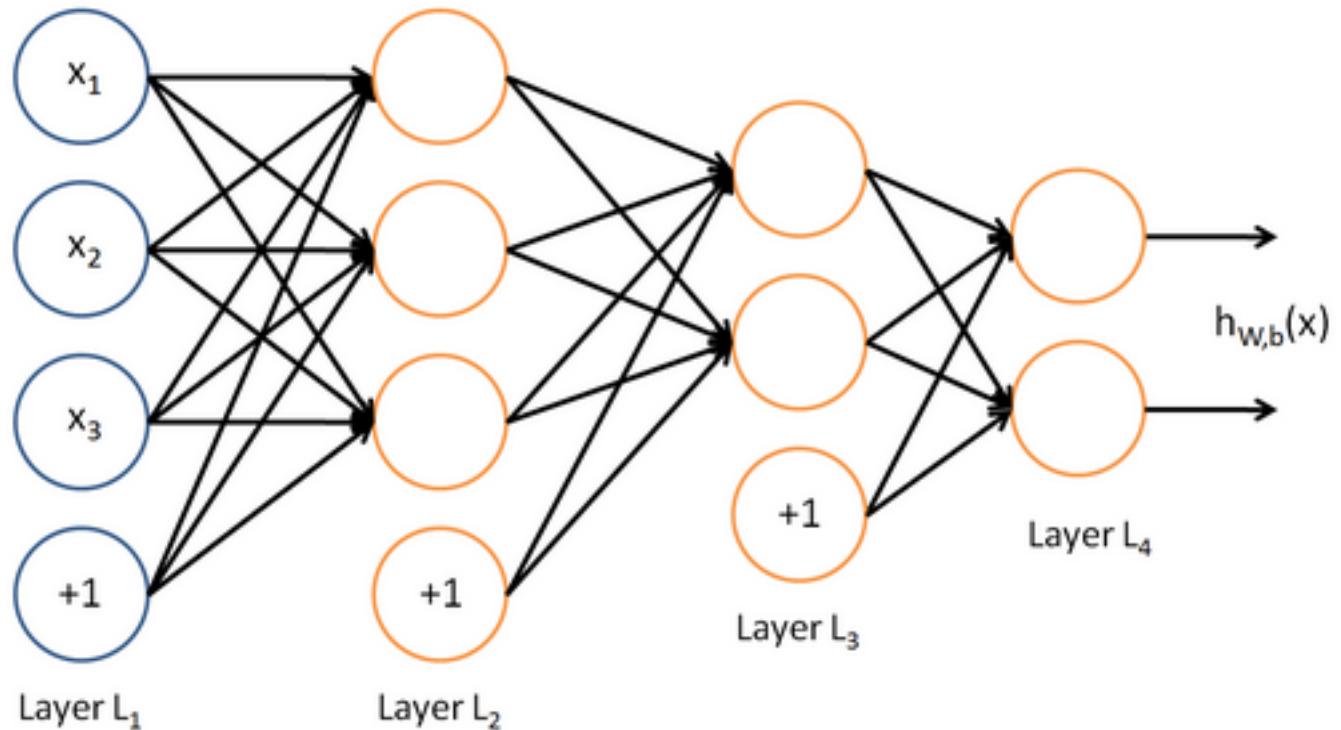
... which we can feed into another logistic regression function



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

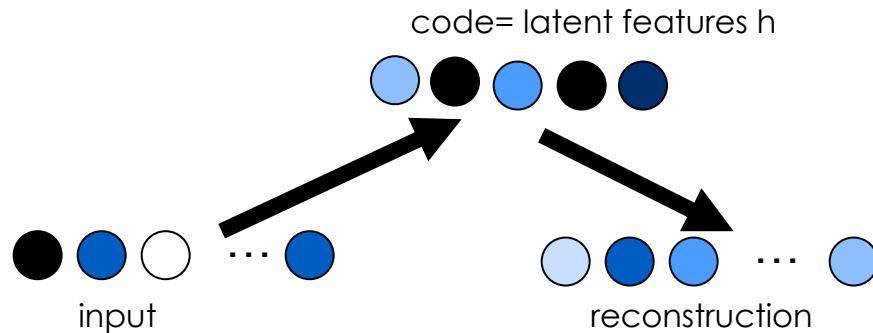
- Before we know it, we have a multilayer neural network....



How to do unsupervised training?

PCA

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

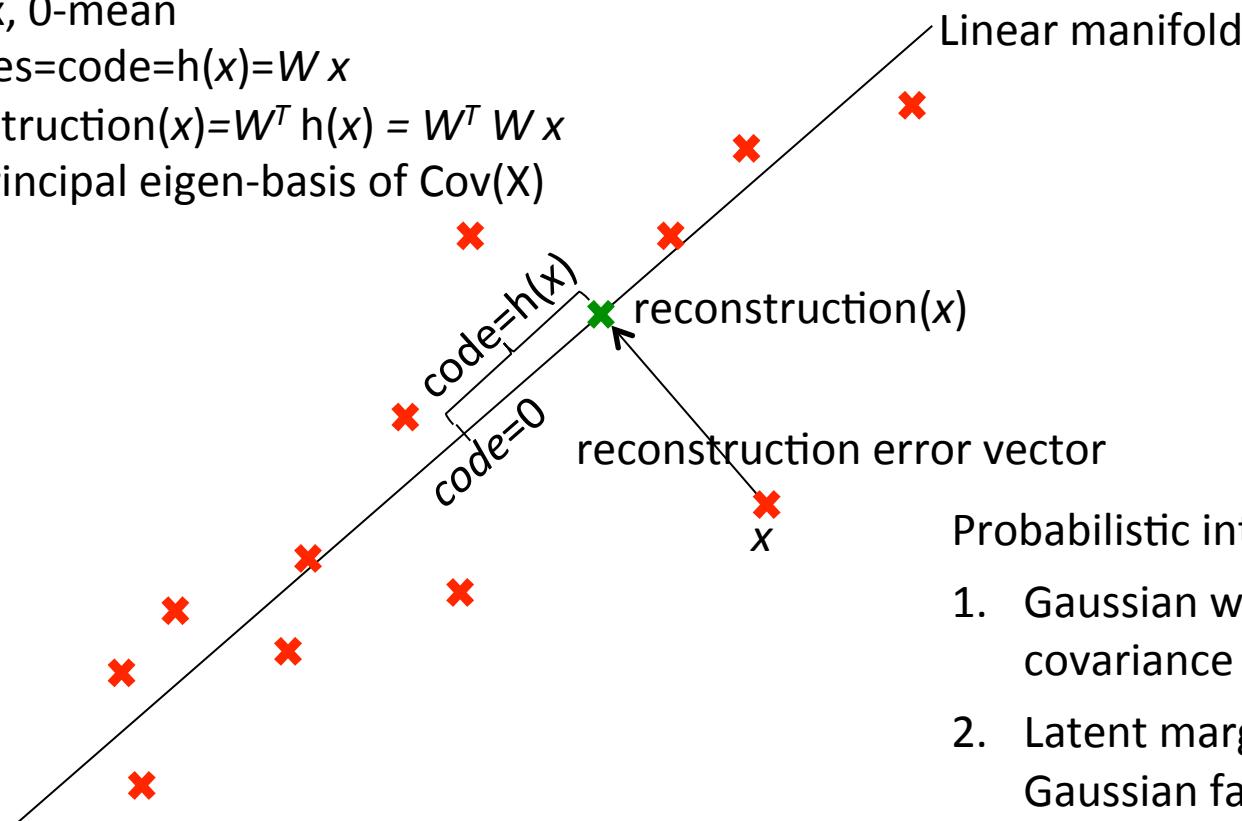


input x , 0-mean

features=code= $h(x)=Wx$

reconstruction(x)= $W^T h(x) = W^T W x$

W = principal eigen-basis of $\text{Cov}(X)$

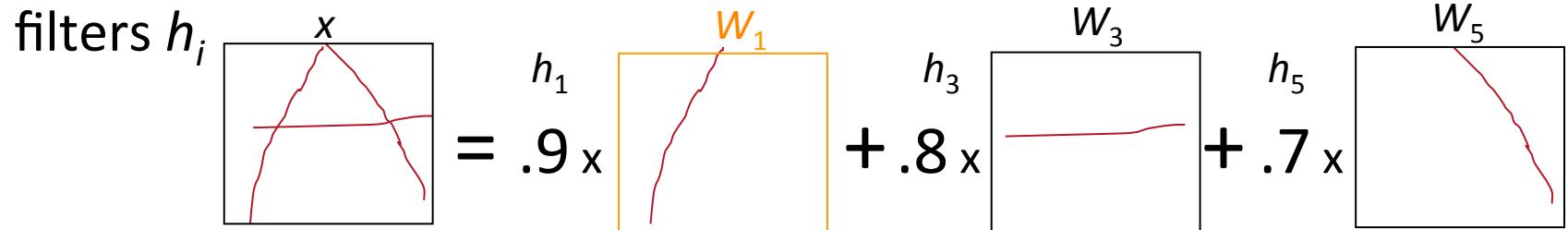
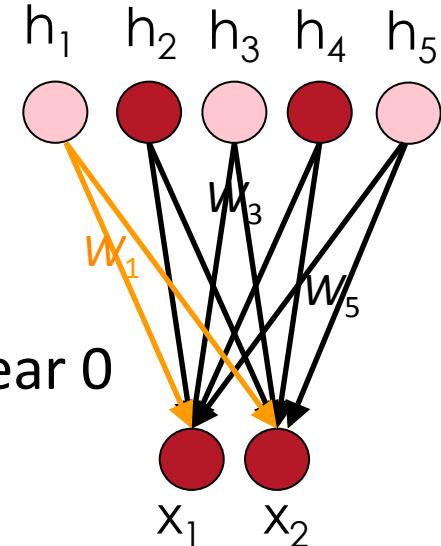


Probabilistic interpretations:

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

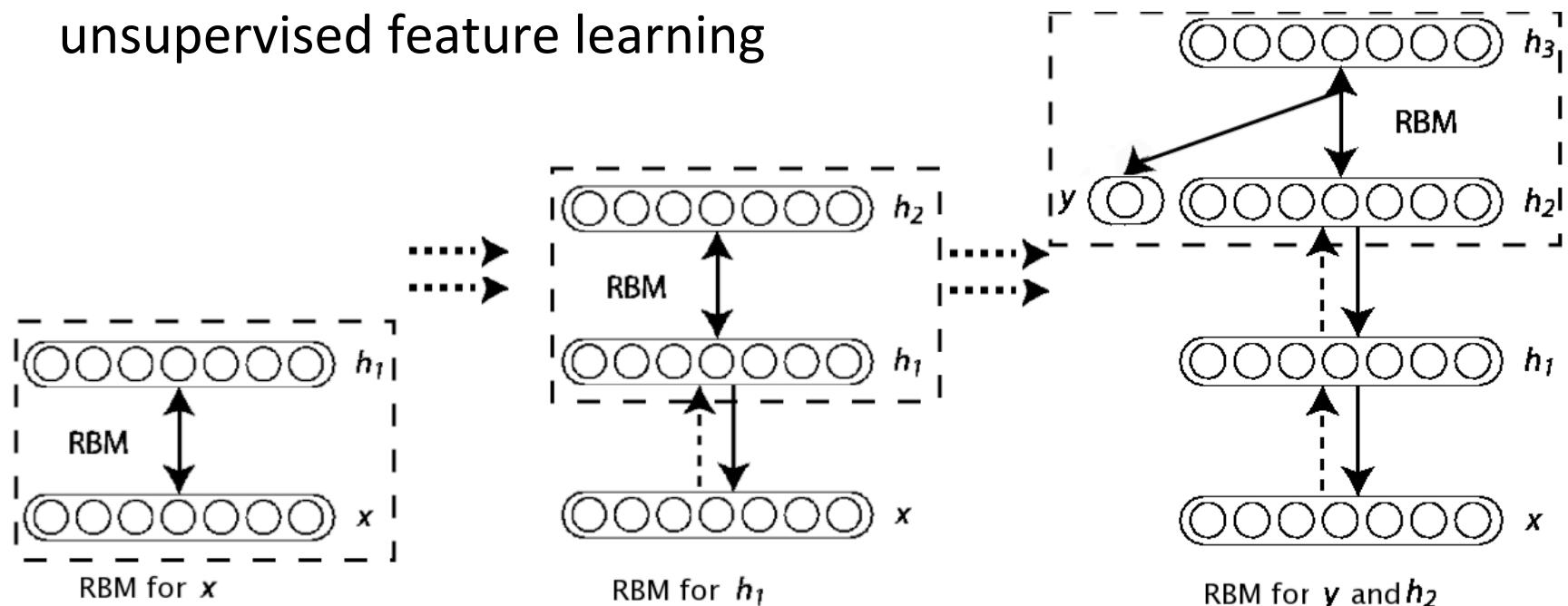
Directed Factor Models

- $P(h)$ factorizes into $P(h_1) P(h_2) \dots$
- 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



Stacking Single-Layer Learners

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



Stacking Restricted Boltzmann Machines (RBM) \rightarrow Deep Belief Network (DBN)

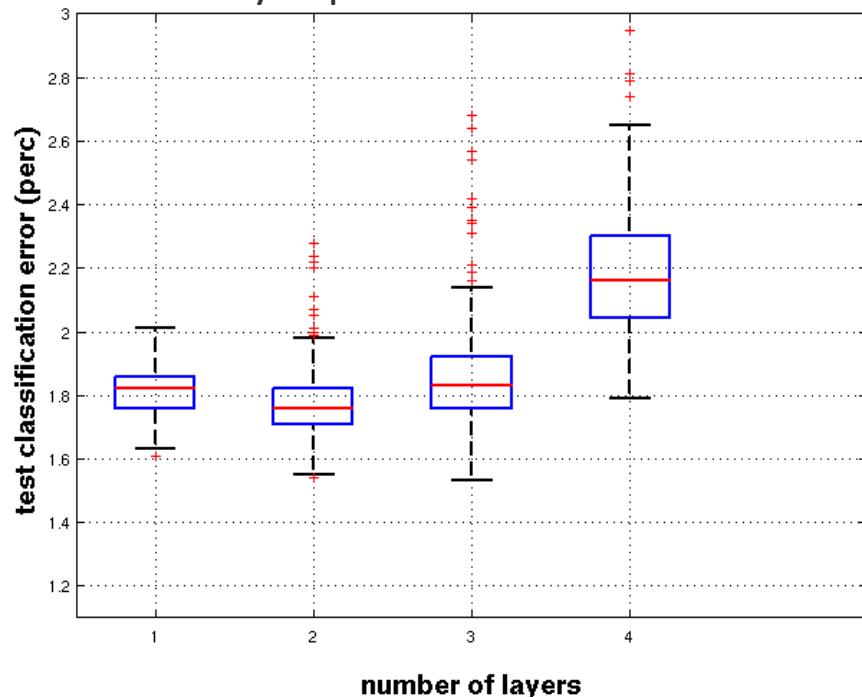
Effective deep Learning became possible through unsupervised pre-training

[Erhan et al., JMLR 2010]

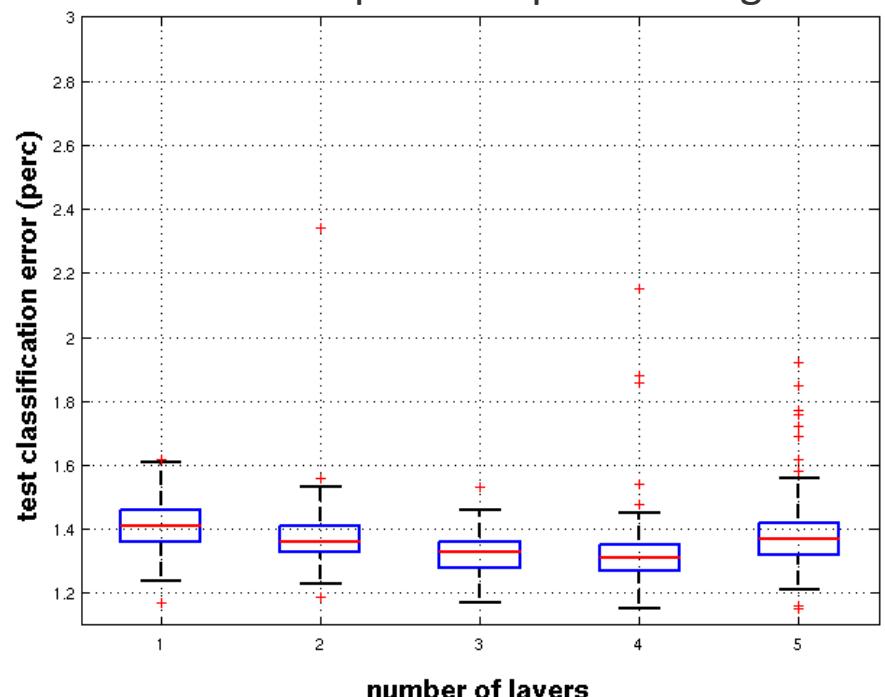


(with RBMs and Denoising Auto-Encoders)

Purely supervised neural net



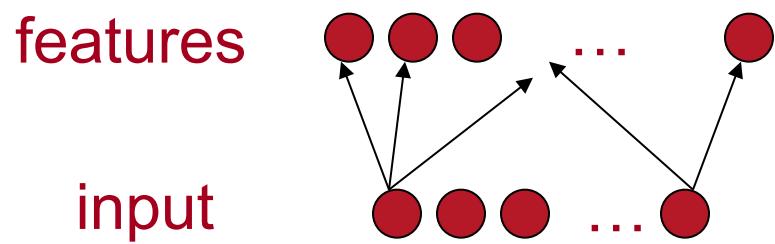
With unsupervised pre-training



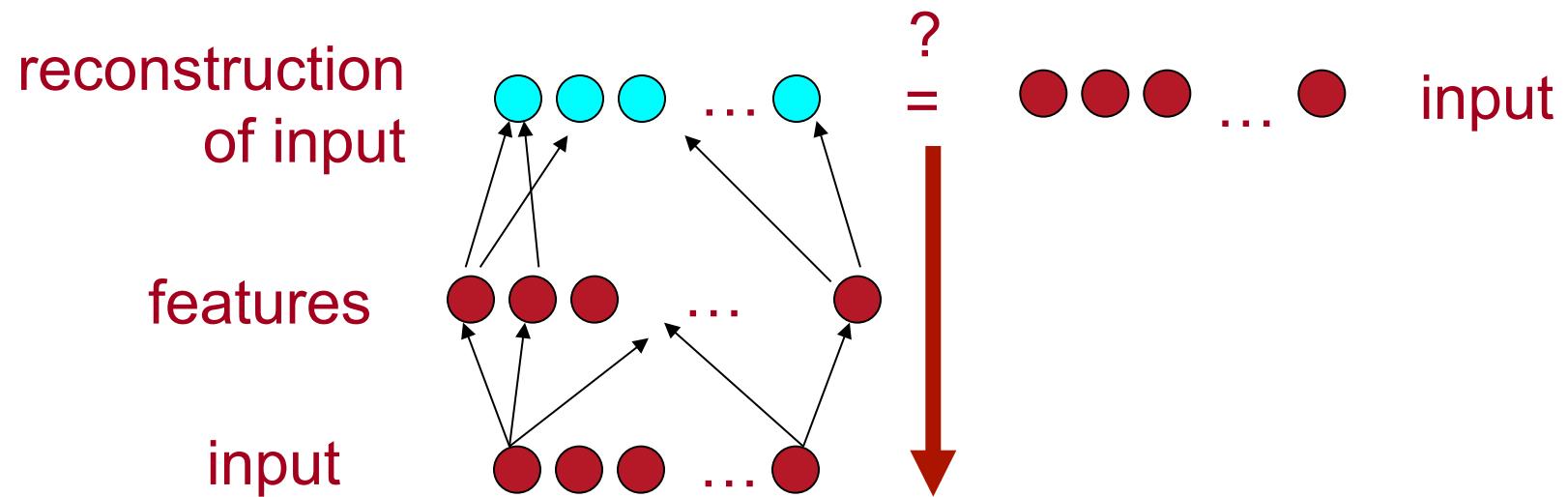
Layer-wise Unsupervised Learning

input 

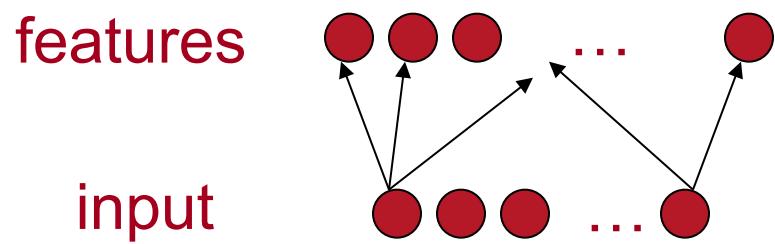
Layer-Wise Unsupervised Pre-training



Layer-Wise Unsupervised Pre-training

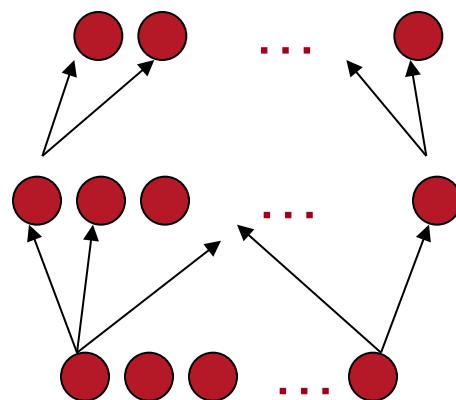


Layer-Wise Unsupervised Pre-training

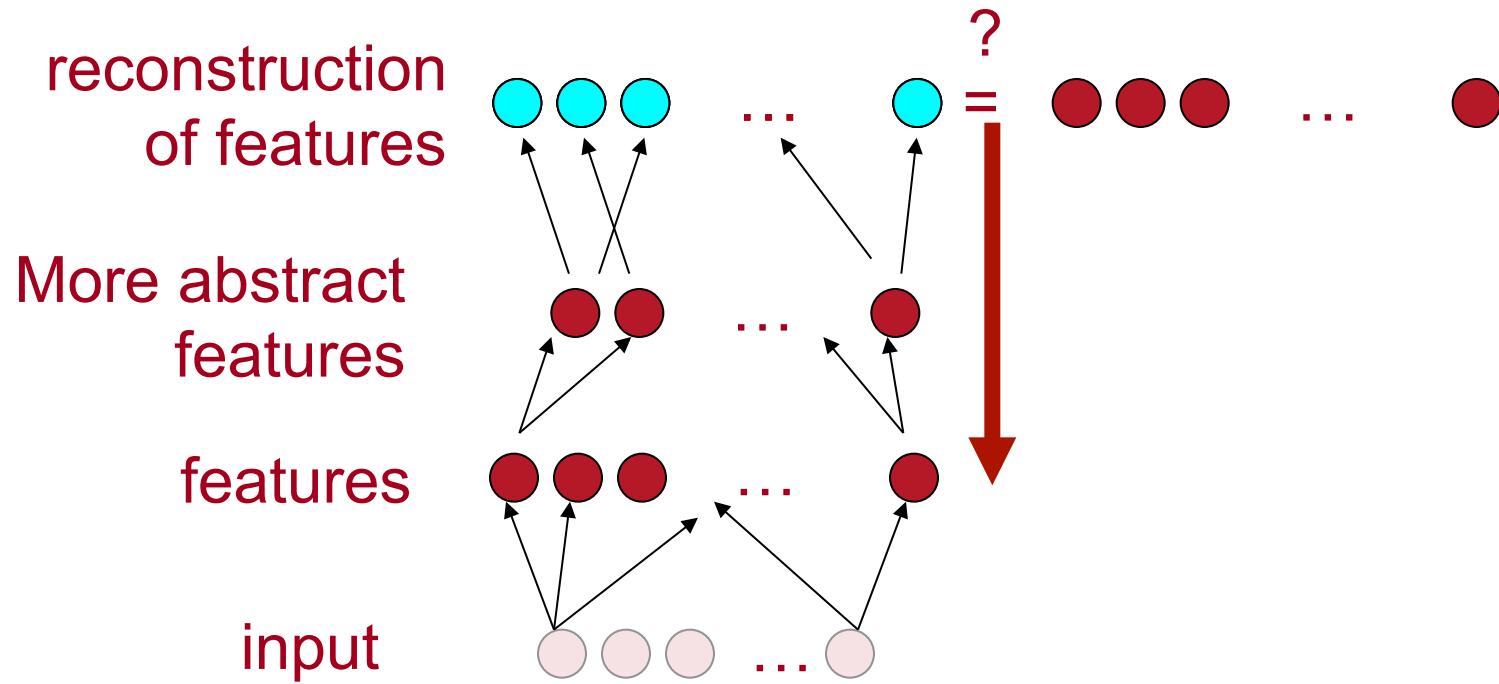


Layer-Wise Unsupervised Pre-training

More abstract
features
features
input

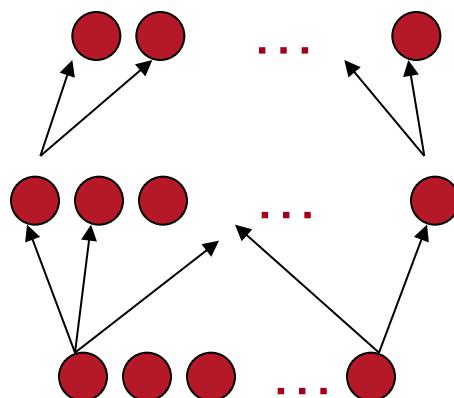


Layer-wise Unsupervised Learning



Layer-Wise Unsupervised Pre-training

More abstract
features
features
input



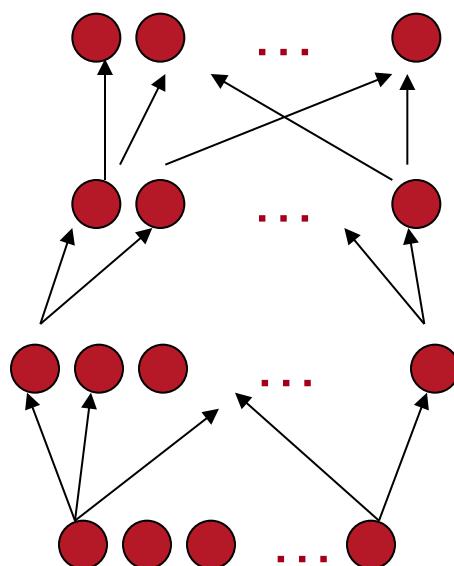
Layer-wise Unsupervised Learning

Even more abstract
features

More abstract
features

features

input



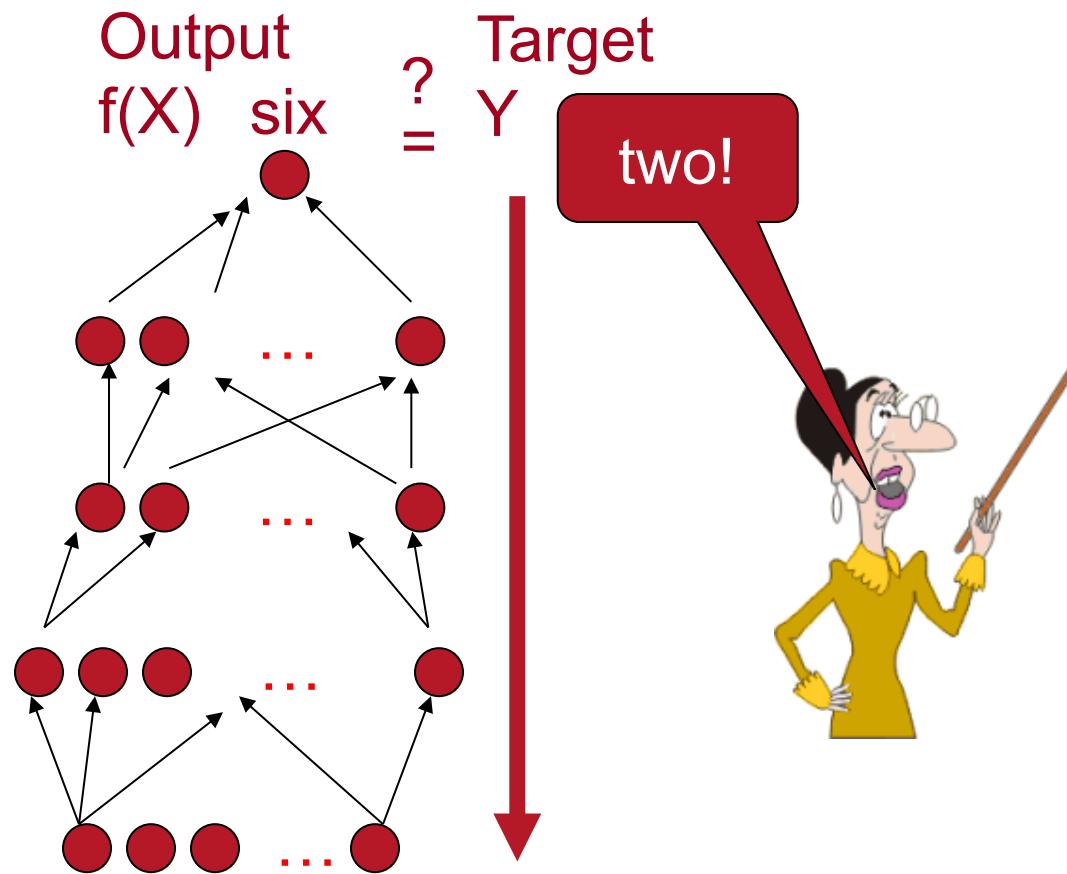
Supervised Fine-Tuning

Even more abstract features

More abstract features

features

input



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

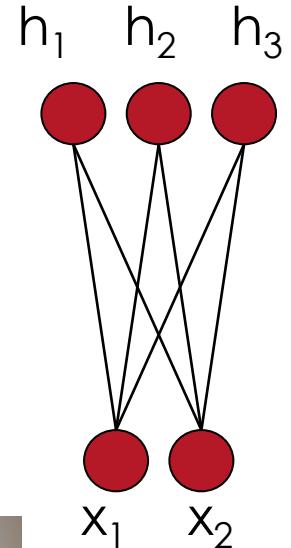
Restricted Boltzmann Machines

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”



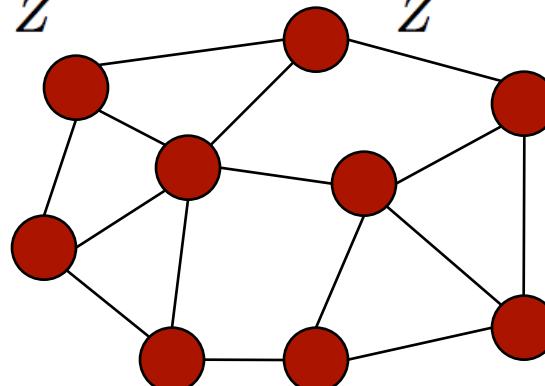
Boltzmann Machines & MRFs

- Boltzmann machines:

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

- Markov Random Fields:

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



Soft constraint / probabilistic statement

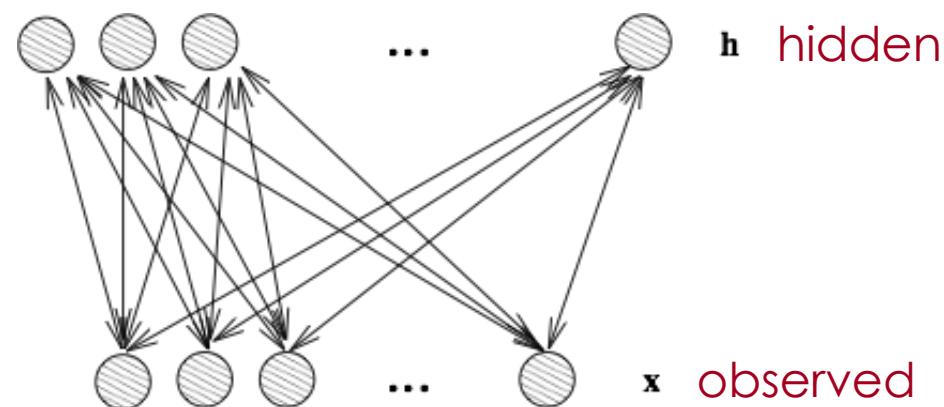
- More interesting with latent variables!

Restricted Boltzmann Machine (RBM)

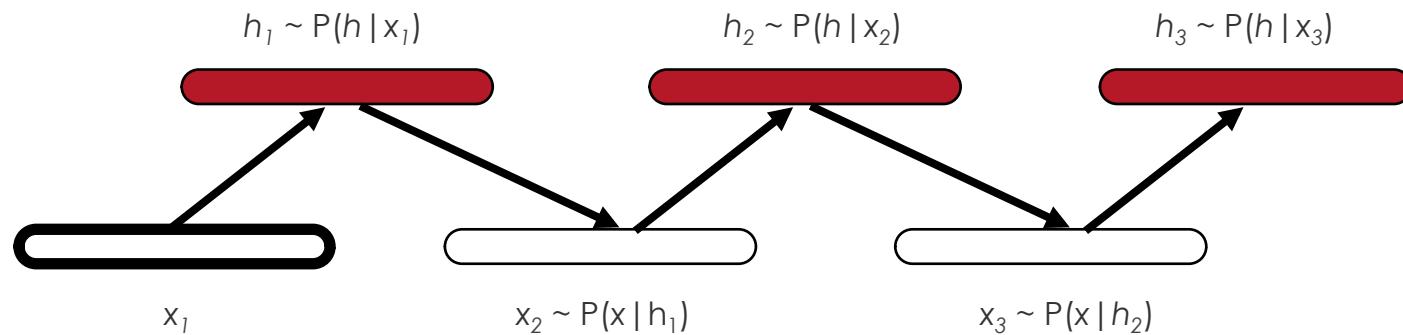
$$P(x, h) = \frac{1}{Z} e^{b^T h + c^T x + h^T W x} = \frac{1}{Z} e^{\sum_i b_i h_i + \sum_j c_j x_j + \sum_{i,j} h_i W_{ij} x_j}$$

- A popular building block for deep architectures

- **Bipartite** undirected graphical model



Gibbs Sampling in RBMs



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

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

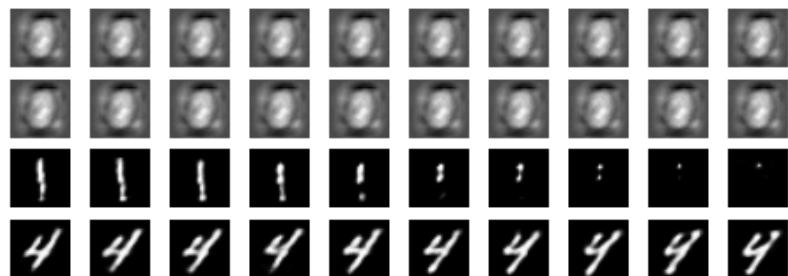
- Easy inference
- Efficient **block Gibbs** sampling $x \rightarrow h \rightarrow x \rightarrow h \dots$

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

Problems with Gibbs Sampling

In practice, Gibbs sampling does not always mix well...

RBM trained by CD on MNIST



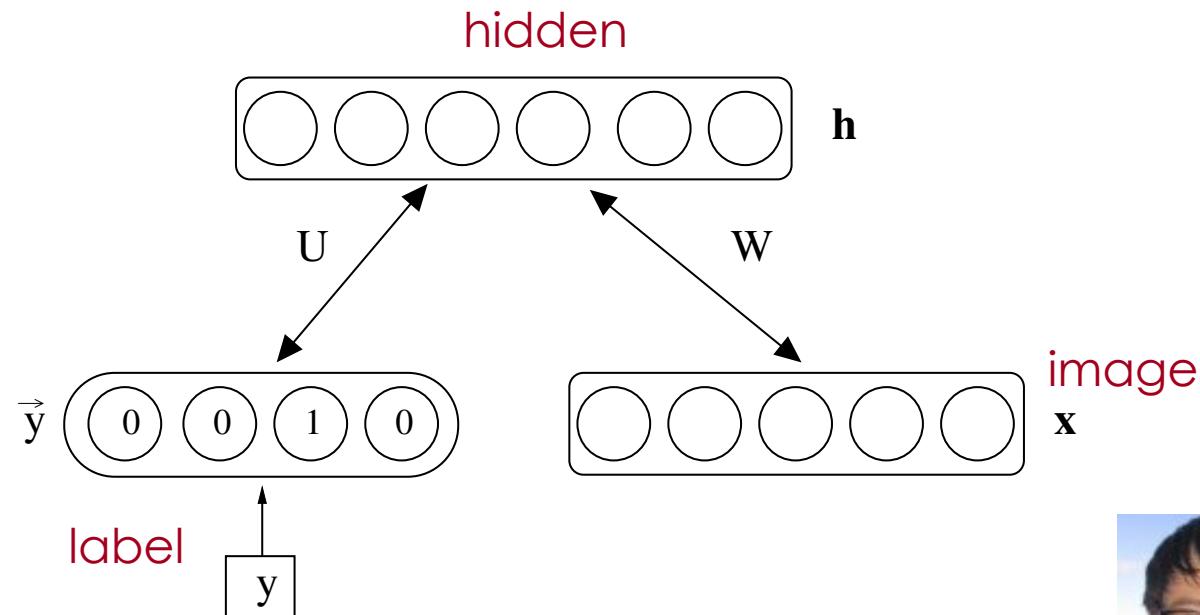
Chains from random state

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 $\text{Energy}(\mathbf{x}, \mathbf{h}) = -\mathbf{b}'\mathbf{x} - \mathbf{c}'\mathbf{h} - \mathbf{h}'W\mathbf{x}$

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

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

RBM 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:

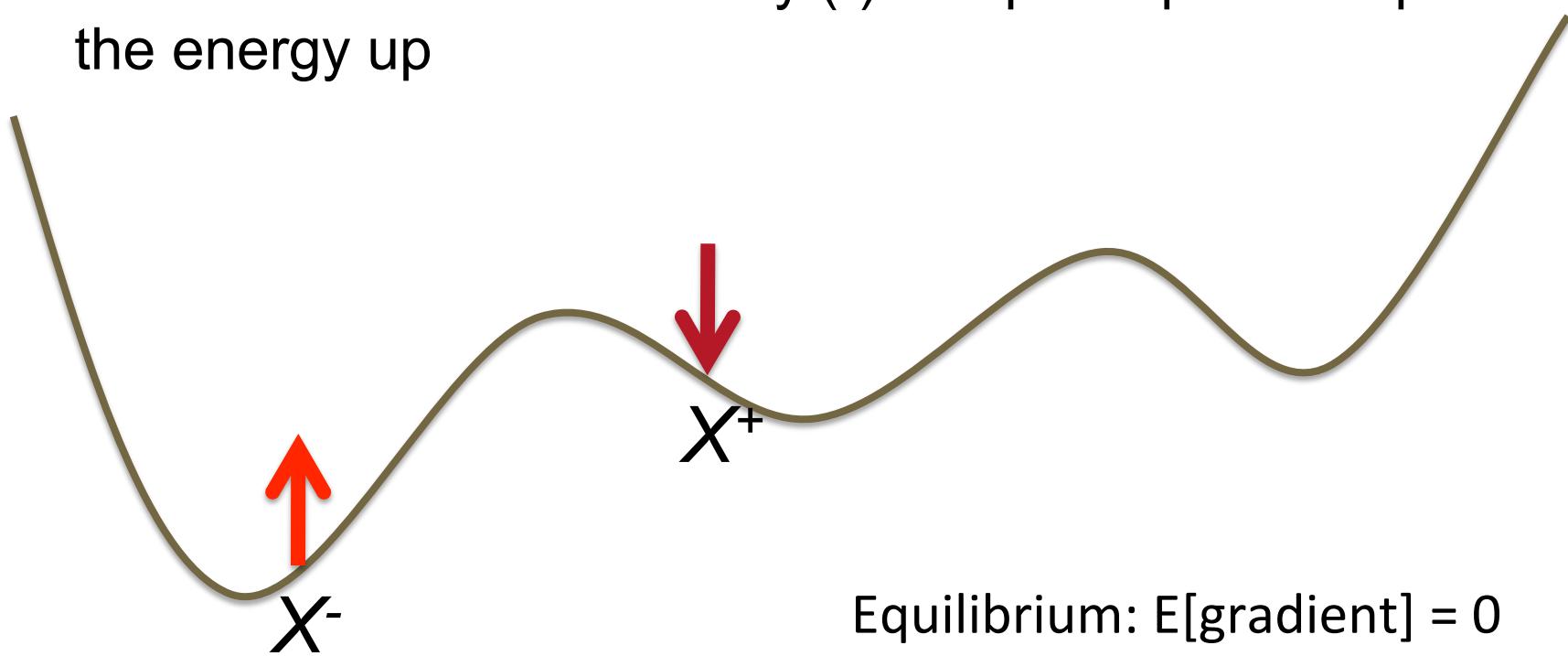
$$\begin{aligned}\frac{\partial \log P(x)}{\partial \theta} &= - \frac{\partial \text{FreeEnergy}(x)}{\partial \theta} + \sum_{\tilde{x}} P(\tilde{x}) \frac{\partial \text{FreeEnergy}(\tilde{x})}{\partial \theta} \\ &= - \sum_h P(h|x) \frac{\partial \text{Energy}(x,h)}{\partial \theta} + \sum_{\tilde{x}, \tilde{h}} P(\tilde{x}, \tilde{h}) \frac{\partial \text{Energy}(\tilde{x}, \tilde{h})}{\partial \theta}\end{aligned}$$

“positive phase” “negative phase”

- 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



Training RBMs

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

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

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

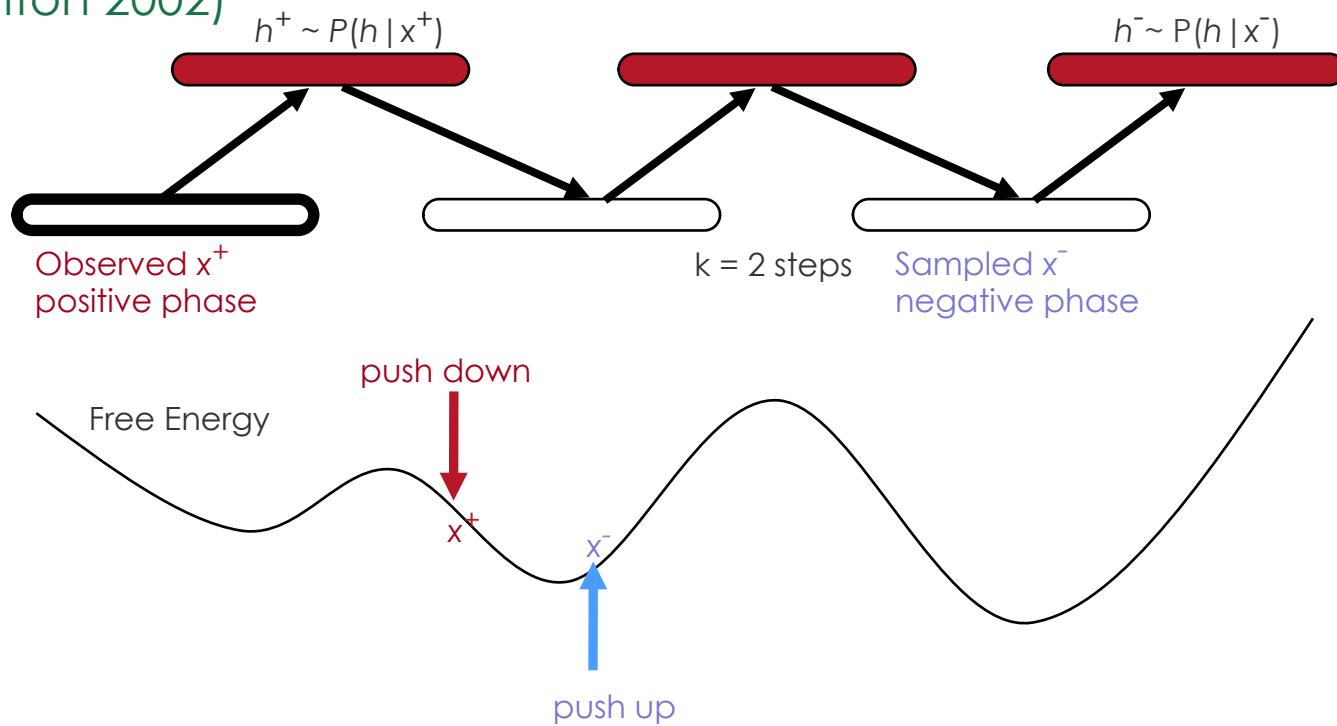
Herding: Deterministic near-chaos dynamical system defines both learning and sampling

Tempered MCMC: use higher temperature to escape modes

Contrastive Divergence



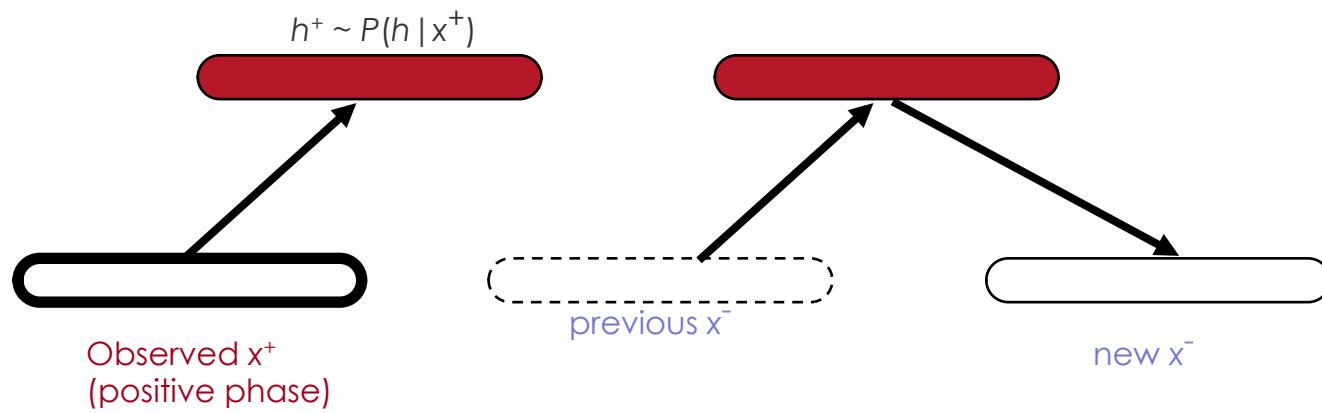
Contrastive Divergence (CD-k): start negative phase
block Gibbs chain at observed x , run k Gibbs steps
(Hinton 2002)



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

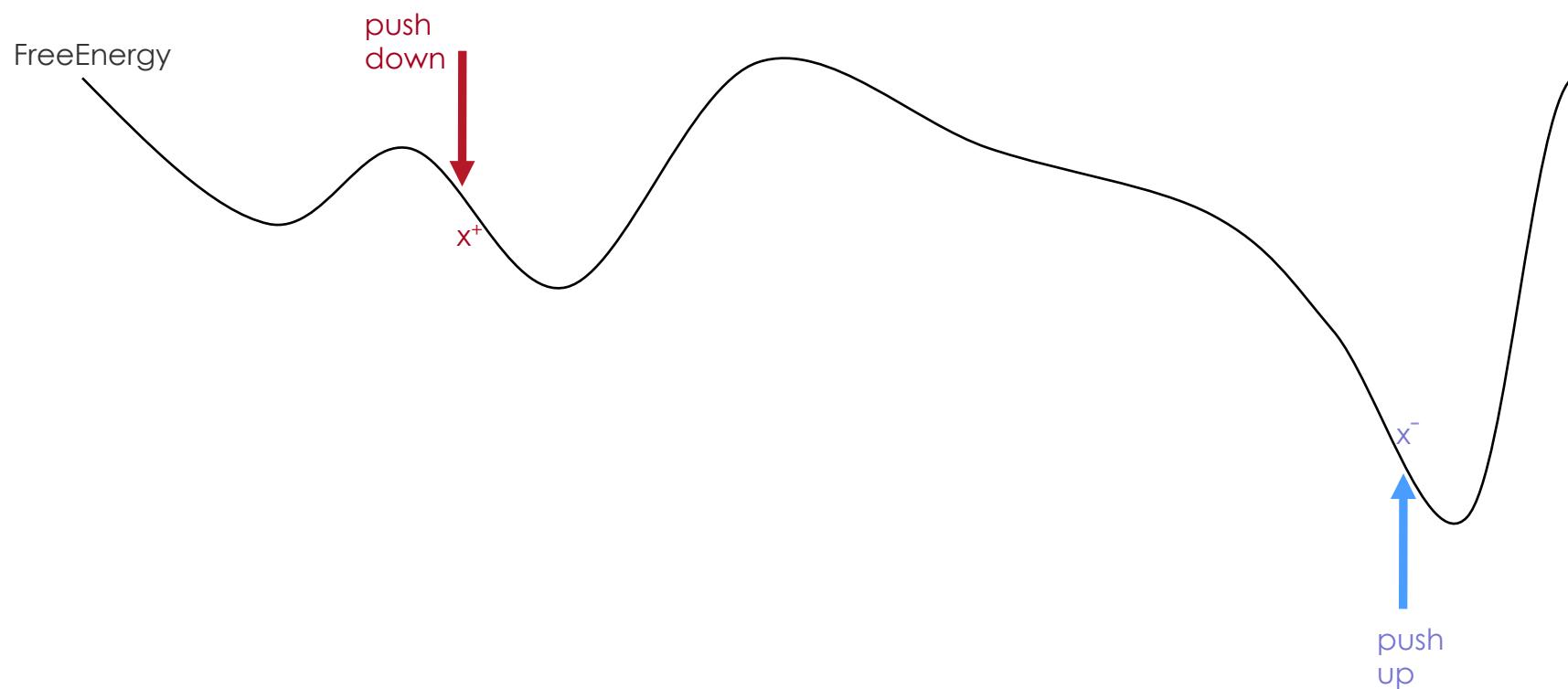
Run negative Gibbs chain in background while weights slowly change (Younes 1999, Tielemans 2008):

- 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

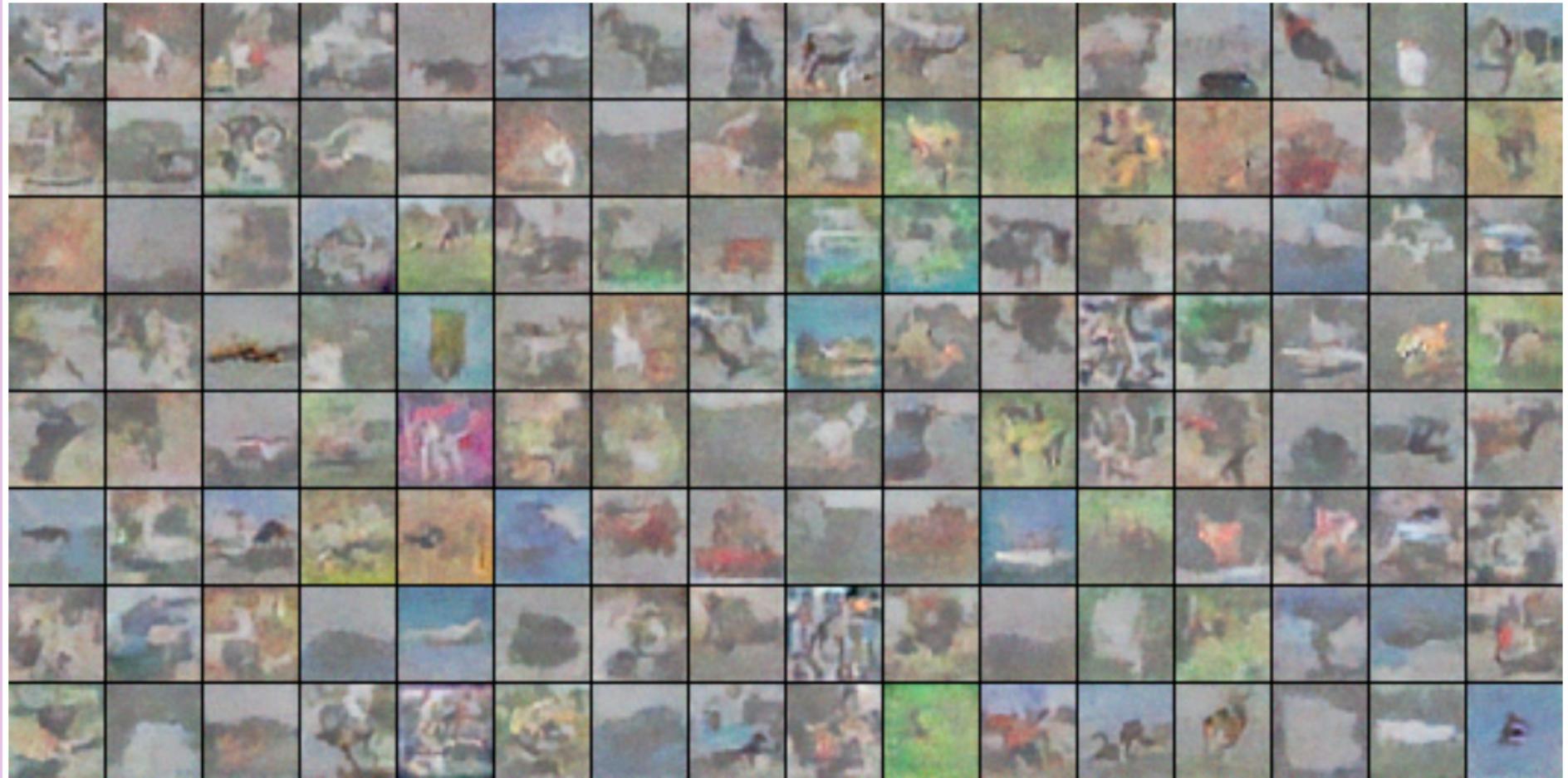


Some RBM Variants

- 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

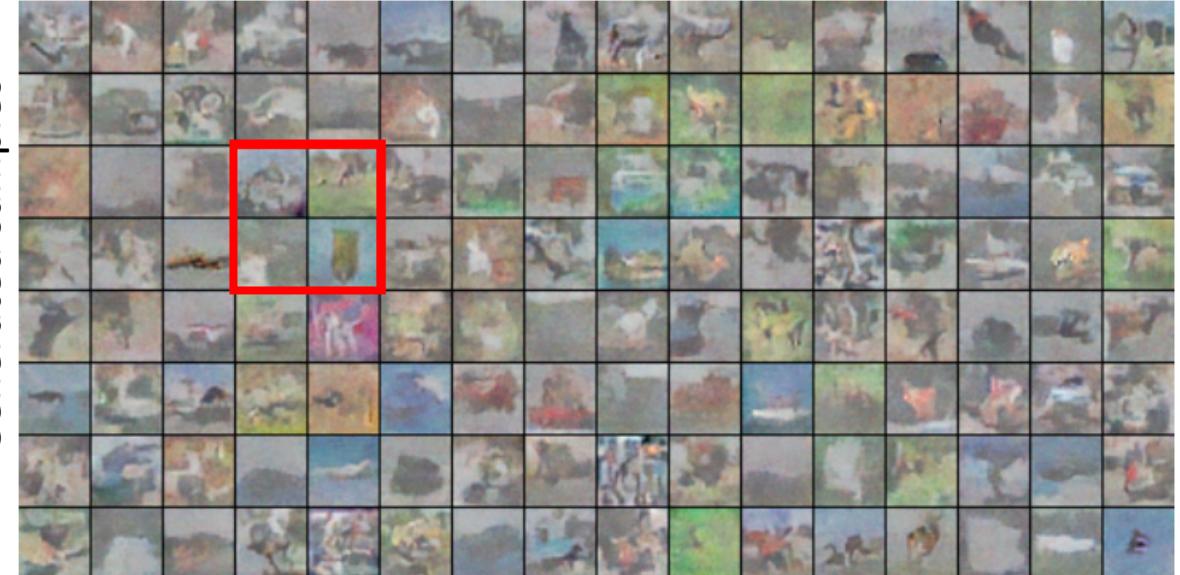
Samples from μ -ssRBM:



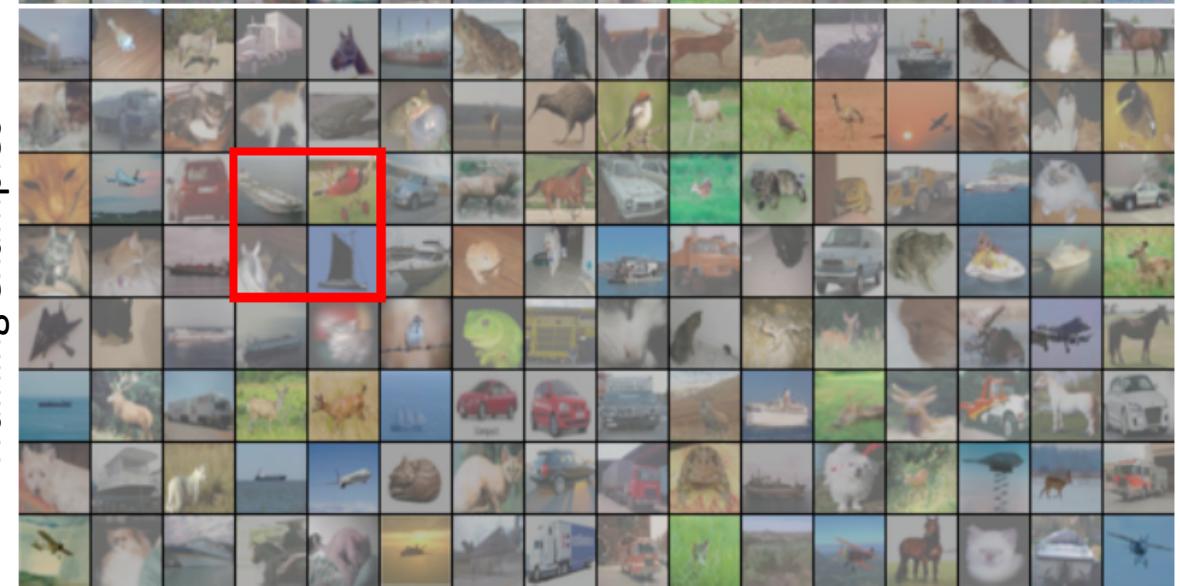
Nearest examples in CIFAR:
(least square dist.)



Generated samples



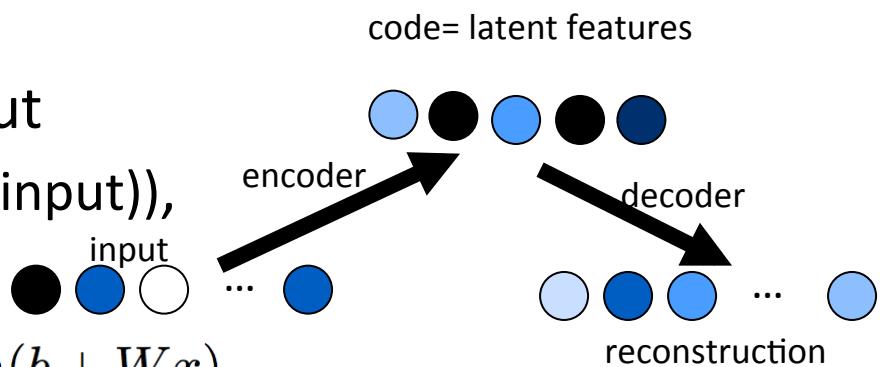
Training examples



Auto-Encoders & Variants

Auto-Encoders

- MLP whose target output = input
- Reconstruction=decoder(encoder(input)),
e.g.



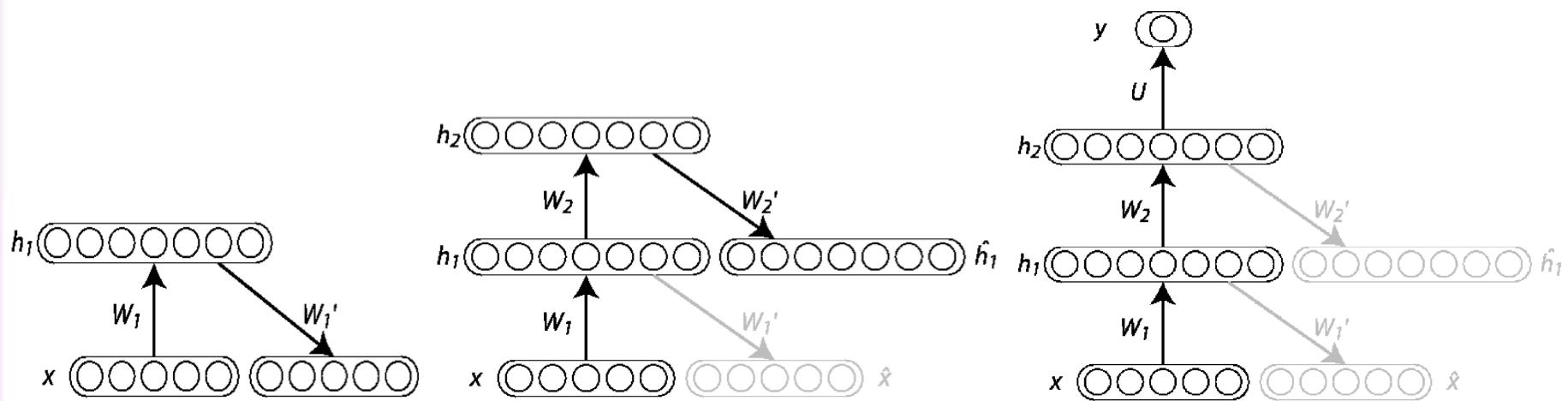
$$h = \tanh(b + Wx)$$

$$\text{reconstruction} = \tanh(c + W^T h)$$

$$\text{Loss } L(x, \text{reconstruction}) = \|\text{reconstruction} - x\|^2$$

- Probable inputs have small reconstruction error because training criterion digs holes at examples
- With bottleneck, code = new coordinate system
- Encoder and decoder can have 1 or more layers
- Training deep auto-encoders notoriously difficult

Stacking Auto-Encoders



Auto-encoders can be stacked successfully (Bengio et al NIPS'2006) to form highly non-linear representations, which with fine-tuning overperformed purely supervised MLPs



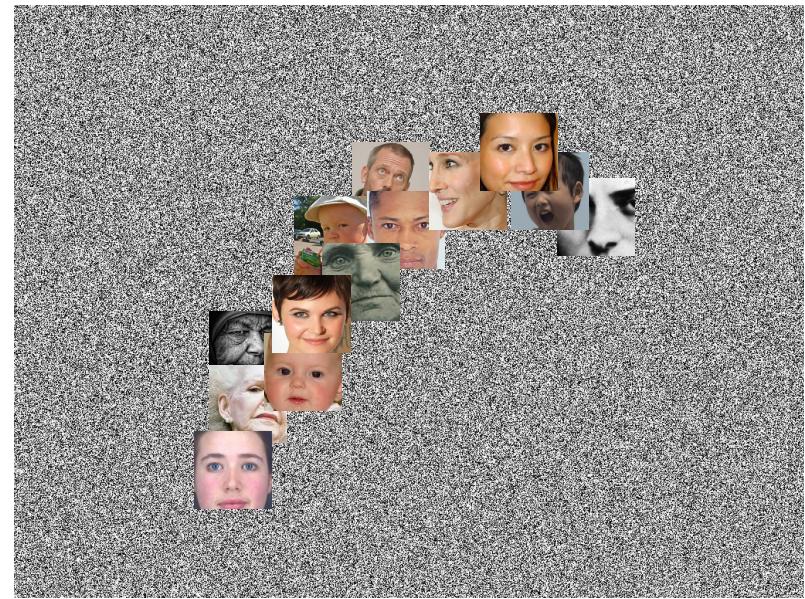
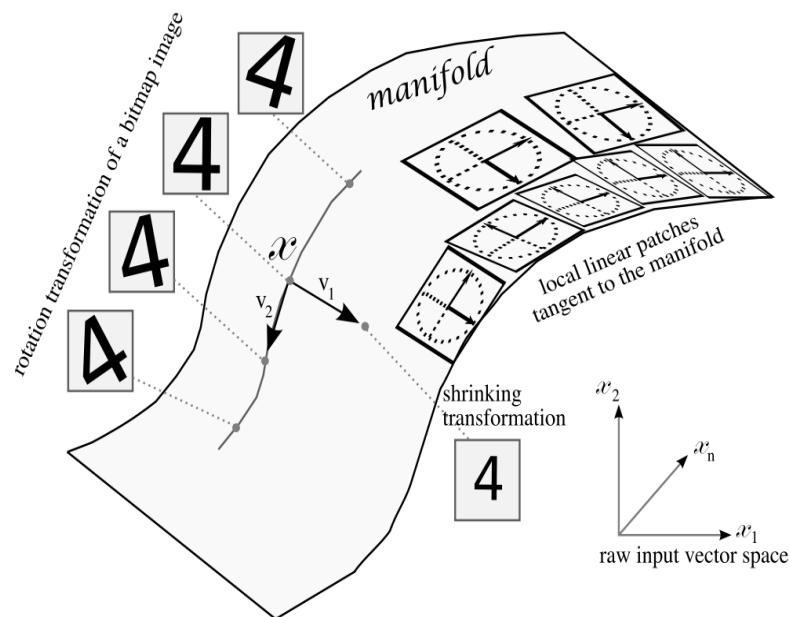
Auto-Encoder Variants

- Discrete inputs: cross-entropy or log-likelihood reconstruction criterion (similar to used for discrete targets for MLPs)
- Regularized to avoid learning the identity everywhere:
 - **Undercomplete** (eg PCA): bottleneck code smaller than input
 - **Sparsity**: encourage hidden units to be at or near 0
[Goodfellow et al 2009]
 - **Denoising**: predict true input from corrupted input
[Vincent et al 2008]
 - **Contractive**: force encoder to have small derivatives
[Rifai et al 2011]



Manifold Learning

- Additional prior: examples concentrate near a lower dimensional “manifold” (region of high density with only few operations allowed which allow small changes while staying on the manifold)

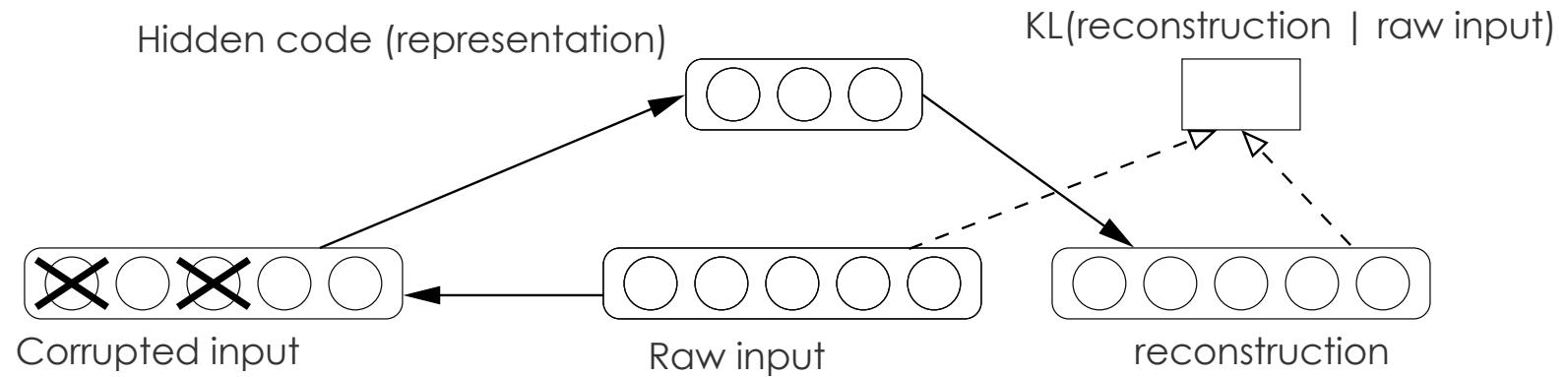


Denoising Auto-Encoder

(Vincent et al 2008)



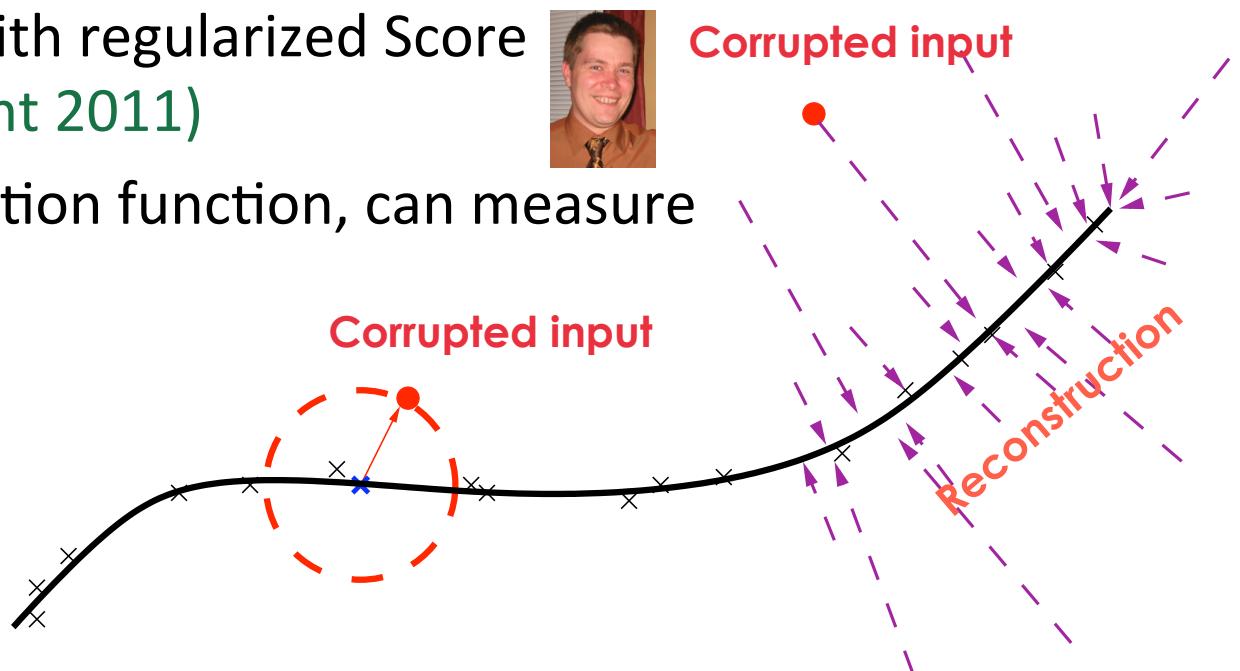
- Corrupt the input
- Reconstruct the uncorrupted input



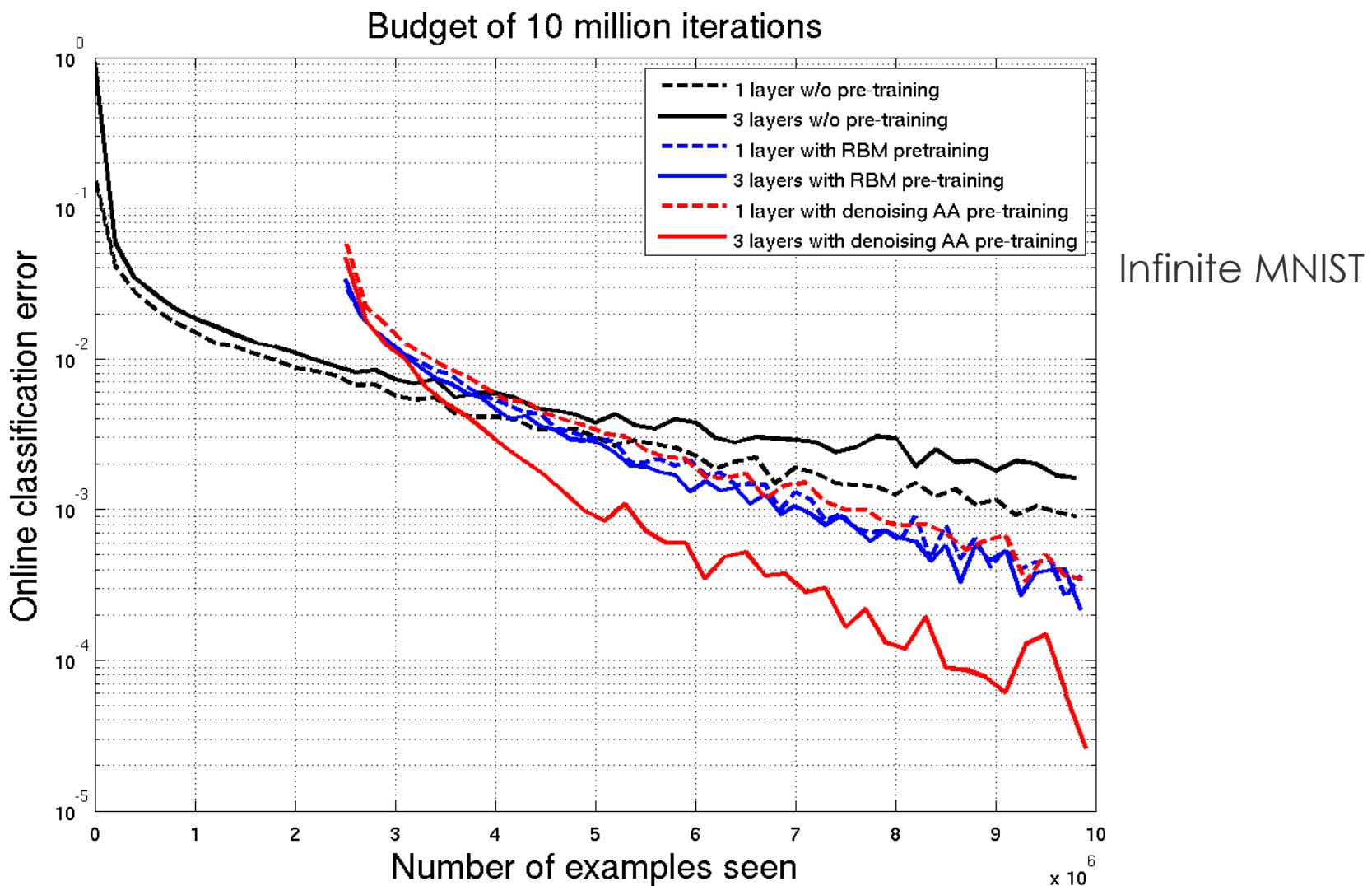
- Encoder & decoder: any parametrization
- As good or better than RBMs for unsupervised pre-training

Denoising Auto-Encoder

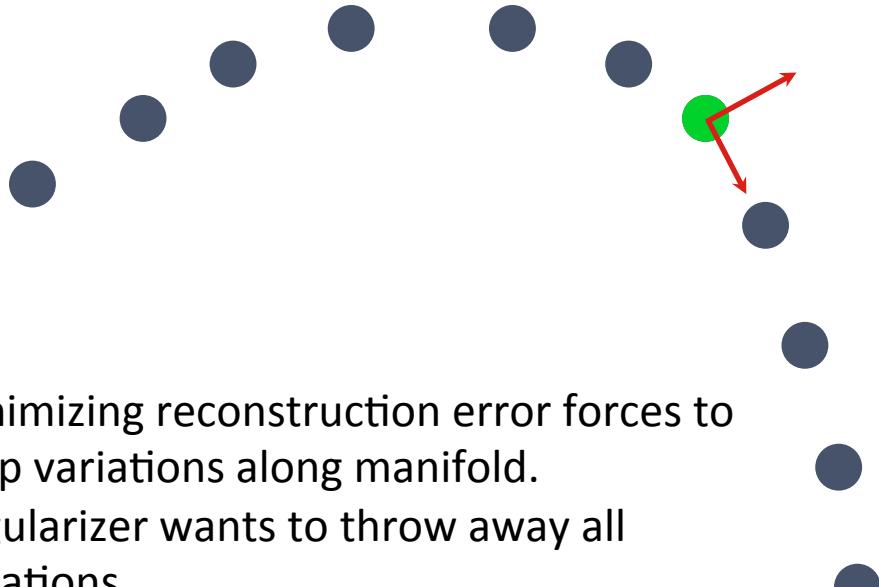
- Learns a vector field towards higher probability regions
- Some DAEs correspond to a kind of Gaussian RBM with regularized Score Matching (Vincent 2011)
- But with no partition function, can measure training criterion



Stacked Denoising Auto-Encoders



Auto-Encoders Learn Salient Variations, Like a non-Linear PCA

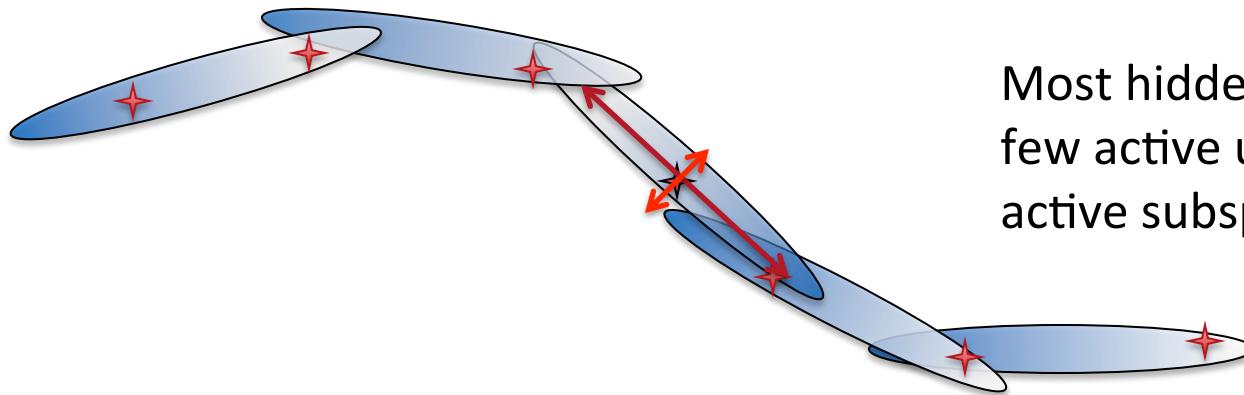


- Minimizing reconstruction error forces to keep variations along manifold.
- Regularizer wants to throw away all variations.
- With both: keep ONLY sensitivity to variations ON the manifold.

Contractive Auto-Encoders



(Rifai, Vincent, Muller, Glorot, Bengio ICML 2011; Rifai, Mesnil, Vincent, Bengio, Dauphin, Glorot ECML 2011; Rifai, Dauphin, Vincent, Bengio, Muller NIPS 2011)



Most hidden units saturate:
few active units represent the
active subspace (local chart)

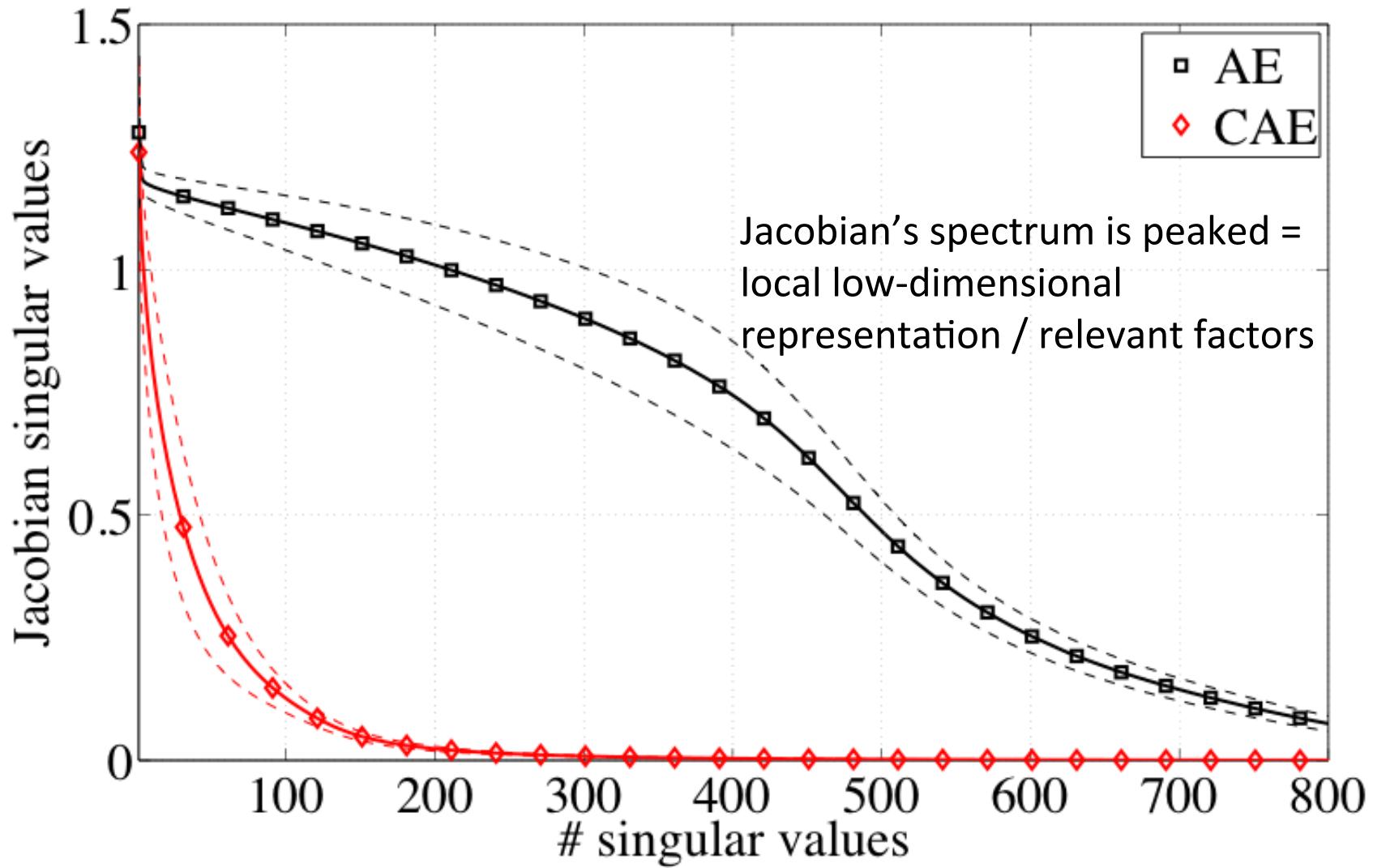
$$\text{reconstruction}(x) = g(h(x)) = \text{decoder}(\text{encoder}(x))$$

$$\mathcal{J}_{CAE}(\theta) = \sum_{x \in D_n} \lambda \sum_{ij} \left(\frac{\partial h_j(x)}{\partial x_i} \right)^2 + L(x, \text{reconstruction}(x))$$

wants contraction in all
directions

cannot afford contraction in
manifold directions

CIFAR-10



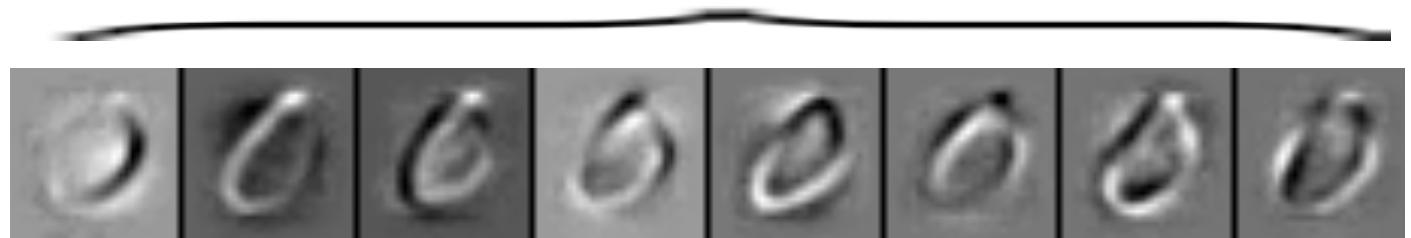
Contractive Auto-Encoders

Data Set	SVM _{rbf}	SAE-3	RBM-3	DAE-b-3	CAE-1	CAE-2
<i>basic</i>	3.03 _{±0.15}	3.46 _{±0.16}	3.11 _{±0.15}	2.84 _{±0.15}	2.83 _{±0.15}	2.48 _{±0.14}
<i>rot</i>	11.11 _{±0.28}	10.30 _{±0.27}	10.30 _{±0.27}	9.53 _{±0.26}	11.59 _{±0.28}	9.66 _{±0.26}
<i>bg-rand</i>	14.58 _{±0.31}	11.28 _{±0.28}	6.73 _{±0.22}	10.30 _{±0.27}	13.57 _{±0.30}	10.90 ±0.27
<i>bg-img</i>	22.61 _{±0.379}	23.00 _{±0.37}	16.31 _{±0.32}	16.68 _{±0.33}	16.70 _{±0.33}	15.50 _{±0.32}
<i>bg-img-rot</i>	55.18 _{±0.44}	51.93 _{±0.44}	47.39 _{±0.44}	43.76 _{±0.43}	48.10 _{±0.44}	45.23 _{±0.44}
<i>rect</i>	2.15 _{±0.13}	2.41 _{±0.13}	2.60 _{±0.14}	1.99 _{±0.12}	1.48 _{±0.10}	1.21 _{±0.10}
<i>rect-img</i>	24.04 _{±0.37}	24.05 _{±0.37}	22.50 _{±0.37}	21.59 _{±0.36}	21.86 _{±0.36}	21.54 _{±0.36}

Input Point



Tangents



$$\text{Input Point} + 0.5 \times \text{Tangents} = \text{Output Image}$$

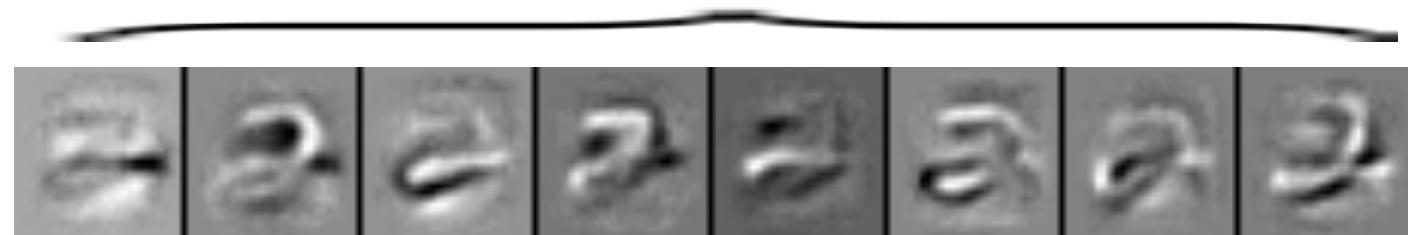
A diagram illustrating a linear transformation. On the left is the 'Input Point' (digit '0'). In the center is the term $+ 0.5 \times$. To its right is the 'Tangents' image, which is being multiplied by 0.5. An equals sign follows, and to its right is the resulting 'Output Image' (digit '0'), which is a slightly smoothed version of the input.

MNIST

Input Point

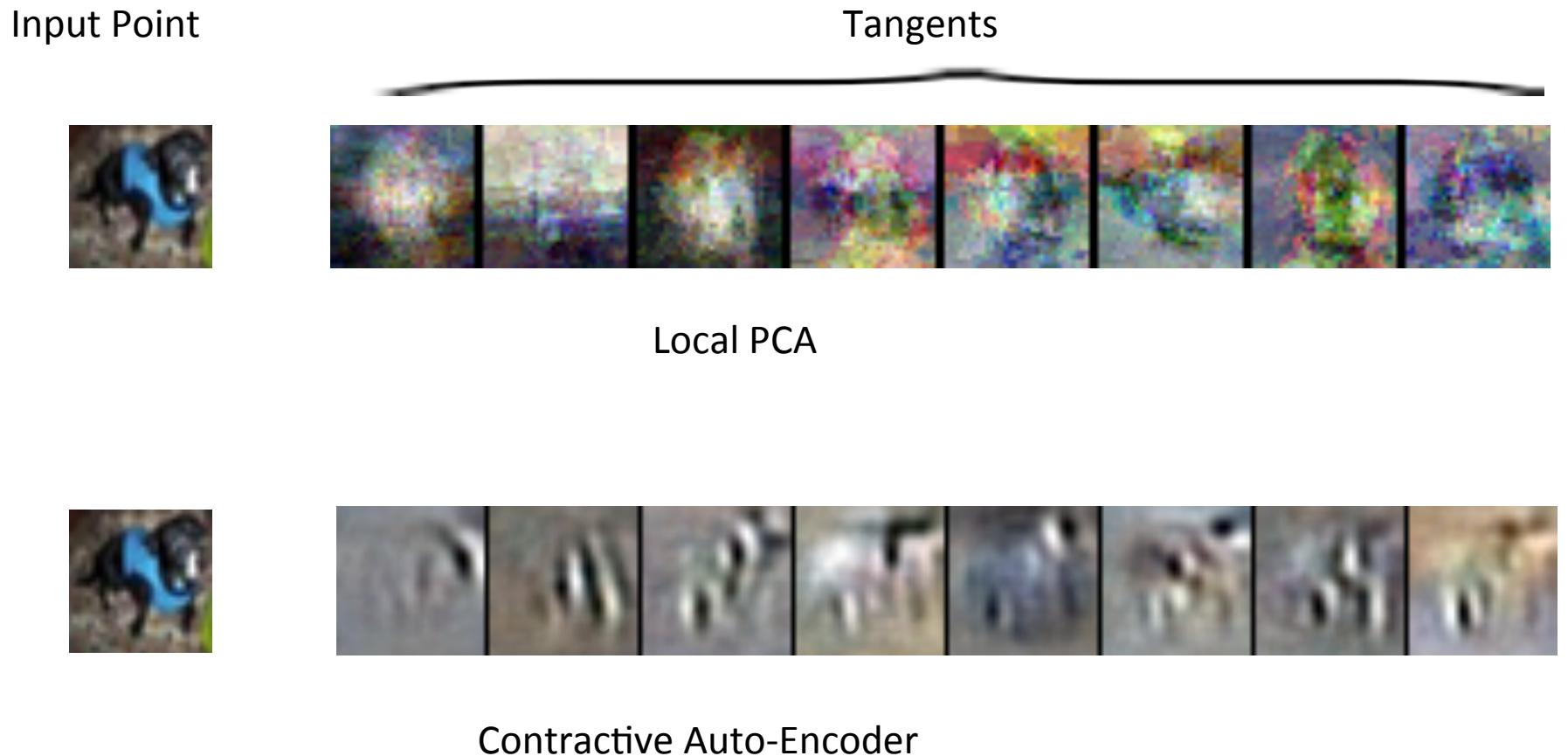


Tangents



MNIST Tangents

Distributed vs Local (CIFAR-10 unsupervised)



Learned Tangent Prop: the Manifold Tangent Classifier

3 hypotheses:

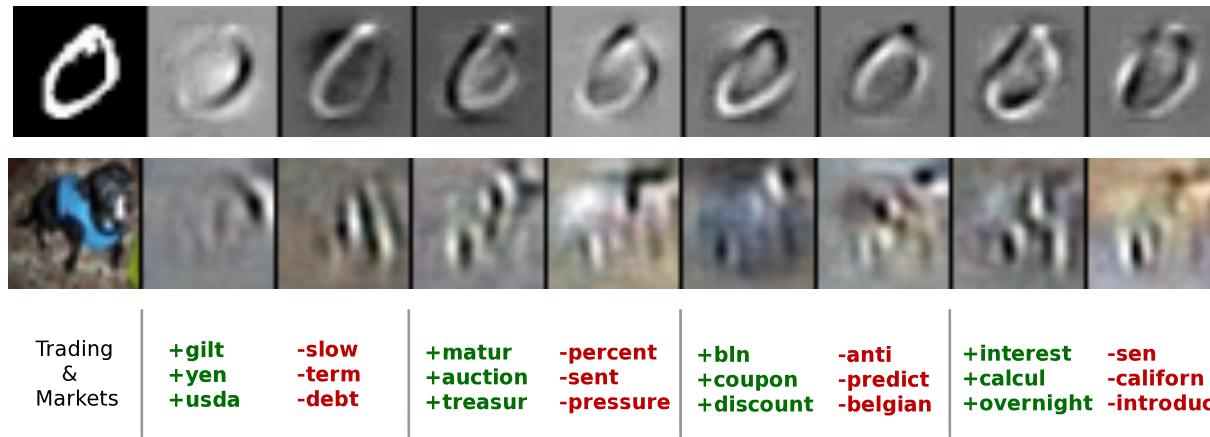
1. Semi-supervised hypothesis ($P(x)$ related to $P(y|x)$)
2. Unsupervised manifold hypothesis (data concentrates near low-dim. manifolds)
3. Manifold hypothesis for classification (low density between class manifolds)

Algorithm:

1. Estimate local principal directions of variation $U(x)$ by CAE (principal singular vectors of $dh(x)/dx$)
2. Penalize $f(x)=P(y|x)$ predictor by $\| df/dx U(x) \|$

Manifold Tangent Classifier Results

- Leading singular vectors on MNIST, CIFAR-10, RCV1:



- Knowledge-free MNIST: 0.81% error

K-NN	NN	SVM	DBN	CAE	DBM	CNN	MTC
3.09%	1.60%	1.40%	1.17%	1.04%	0.95%	0.95%	0.81%

- Semi-sup.

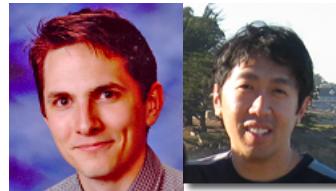
	NN	SVM	CNN	TSVM	DBN-rNCA	EmbedNN	CAE	MTC
100	25.81	23.44	22.98	16.81	-	16.86	13.47	12.03
600	11.44	8.85	7.68	6.16	8.7	5.97	6.3	5.13
1000	10.7	7.77	6.45	5.38	-	5.73	4.77	3.64
3000	6.04	4.21	3.35	3.45	3.3	3.59	3.22	2.57

- Forest (500k examples)

SVM	Distributed SVM	MTC
4.11%	3.46%	3.13%

Inference and Explaining Away

- Easy inference in RBMs and regularized Auto-Encoders
- But no explaining away (competition between causes)
- (Coates et al 2011): even when training filters as RBMs it helps to perform additional explaining away (e.g. plug them into a Sparse Coding inference), to obtain better-classifying features



- RBMs would need lateral connections to achieve similar effect
- Auto-Encoders would need to have lateral recurrent connections

Sparse Coding

(Olshausen et al 97)



- Directed graphical model:

$$P(h) \propto e^{-\lambda|h|_1} \quad x|h \sim N(W^T h, \sigma^2 I)$$

- One of the first unsupervised feature learning algorithms with non-linear feature extraction (but linear decoder)

$$\min_h \frac{\|x - W^T h\|^2}{\sigma^2} + \lambda|h|_1$$

MAP inference recovers sparse h although $P(h|x)$ not concentrated at 0

- Linear decoder, non-parametric encoder
- Sparse Coding inference, convex opt. but expensive

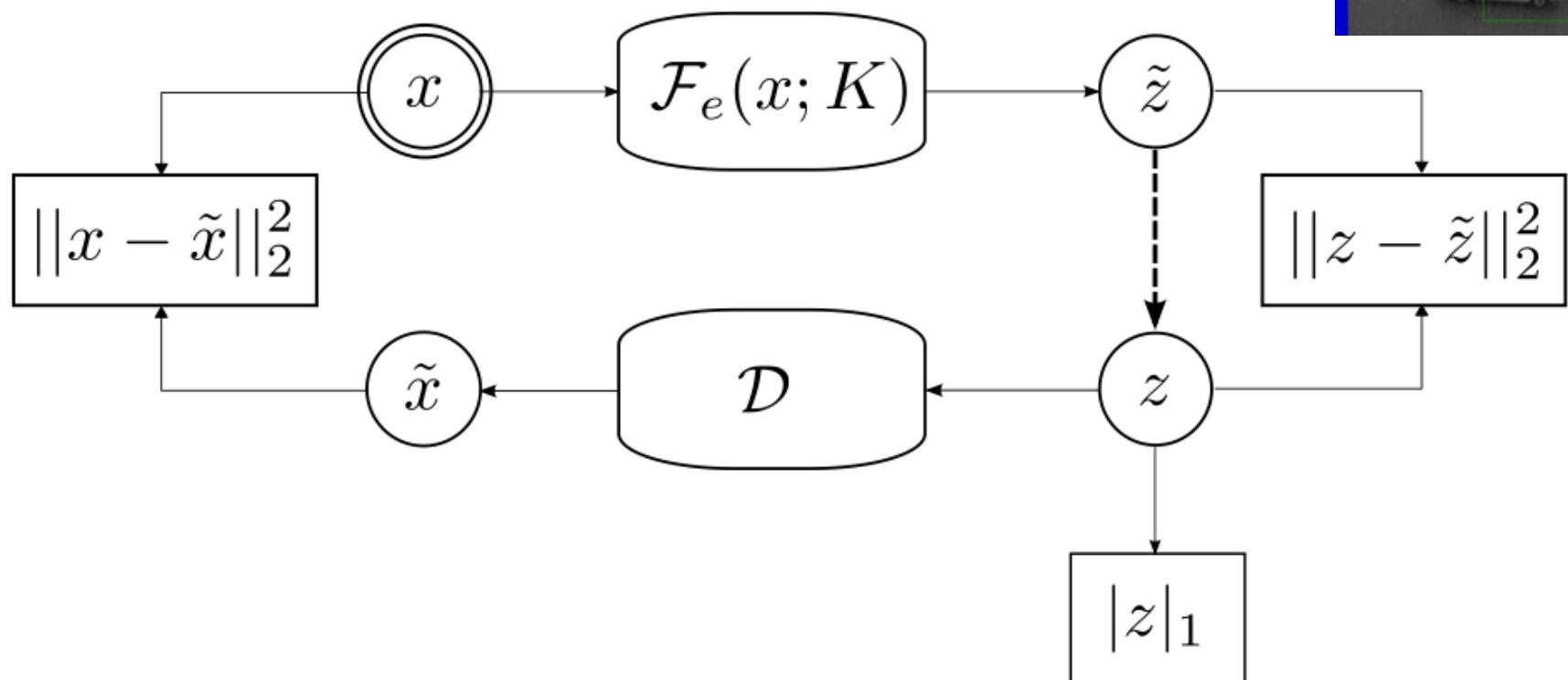
Predictive Sparse Decomposition

- Approximate the inference of sparse coding by an encoder:



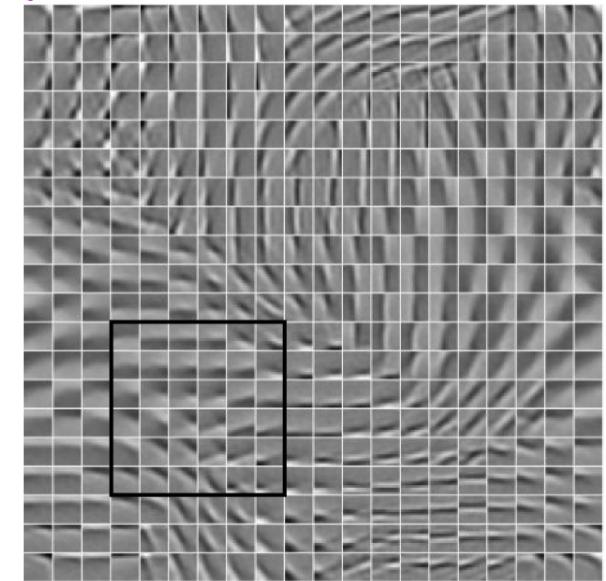
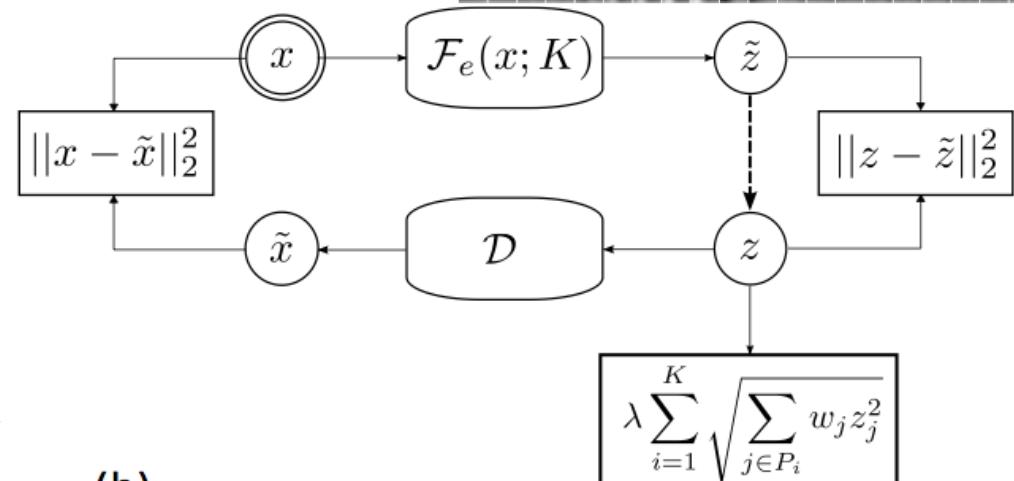
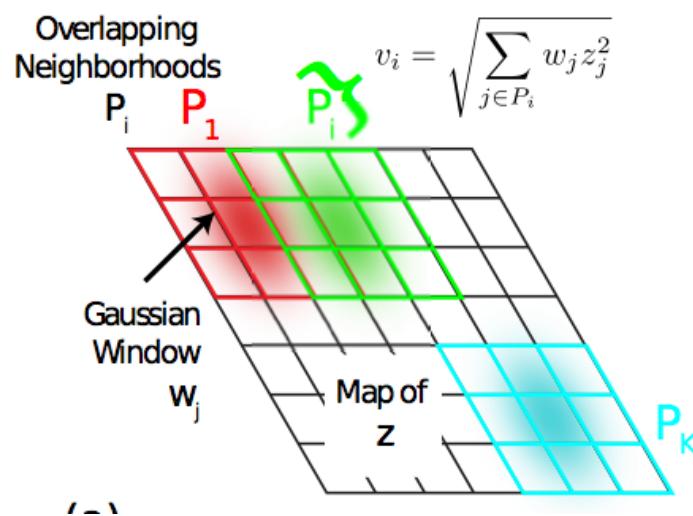
Predictive Sparse Decomposition (Kavukcuoglu et al 2008)

- Very successful applications in machine vision with convolutional architectures



Predictive Sparse Decomposition

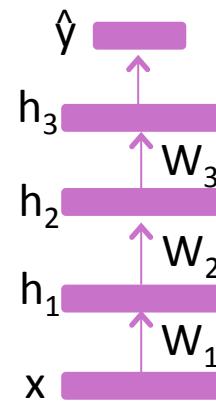
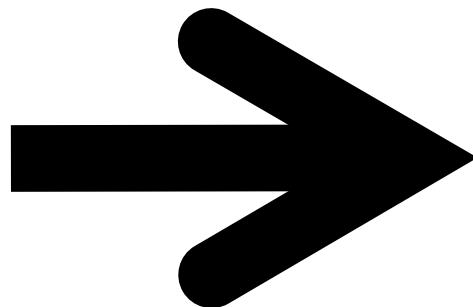
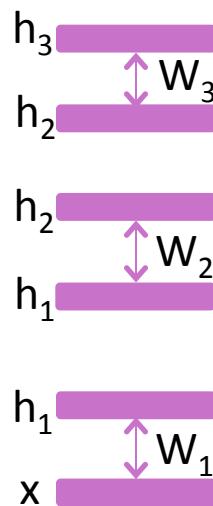
- Stacked to form deep architectures
- Alternating convolution, rectification, pooling
- Tiling: no sharing across overlapping filters
- Group sparsity penalty yields topographic maps



Deep Variants

Stack of RBMs / AEs → Deep MLP

- Encoder or $P(h|v)$ becomes MLP layer

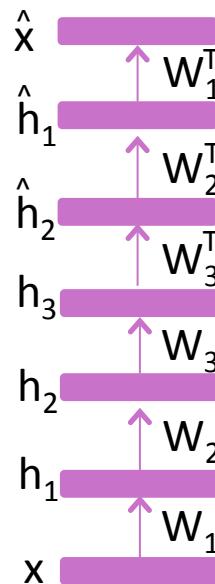
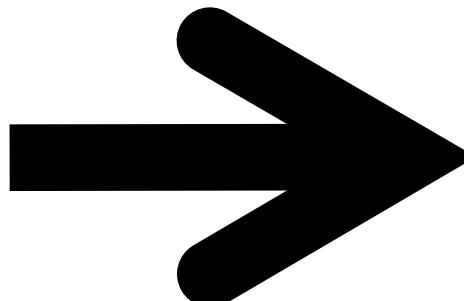
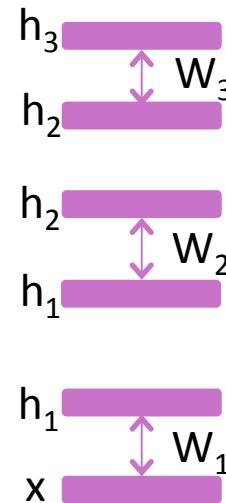


Stack of RBMs / AEs → Deep Auto-Encoder

(Hinton & Salakhutdinov 2006)



- Stack encoders / $P(h|x)$ into deep encoder
- Stack decoders / $P(x|h)$ into deep decoder

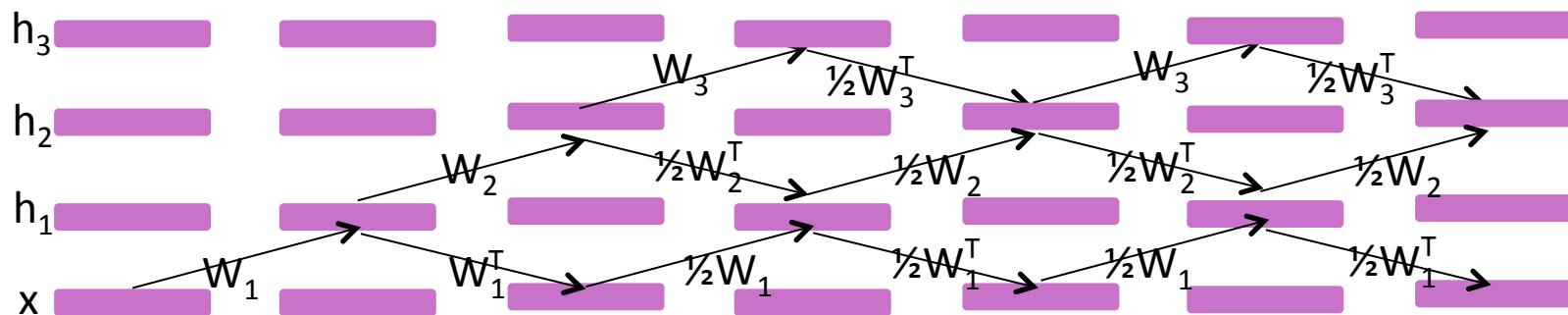
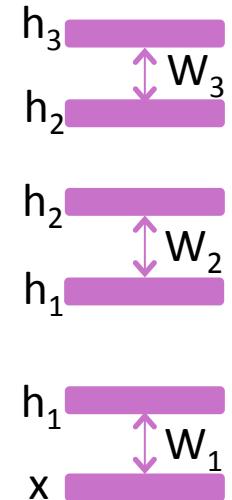


Stack of RBMs / AEs → Deep Recurrent Auto-Encoder

(Savard 2011)



- Each hidden layer receives input from below and above
- Halve the weights
- Deterministic (mean-field) recurrent computation

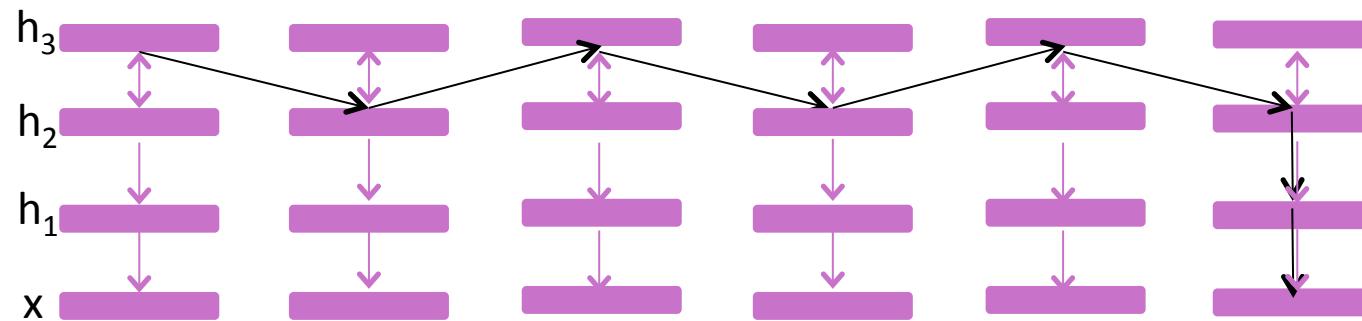


Stack of RBMs → Deep Belief Net



(Hinton et al 2006)

- Stack lower levels RBMs' $P(x|h)$ along with top-level RBM
- $P(x, h_1, h_2, h_3) = P(h_2, h_3) P(h_1|h_2) P(x | h_1)$
- Sample: Gibbs on top RBM, propagate down

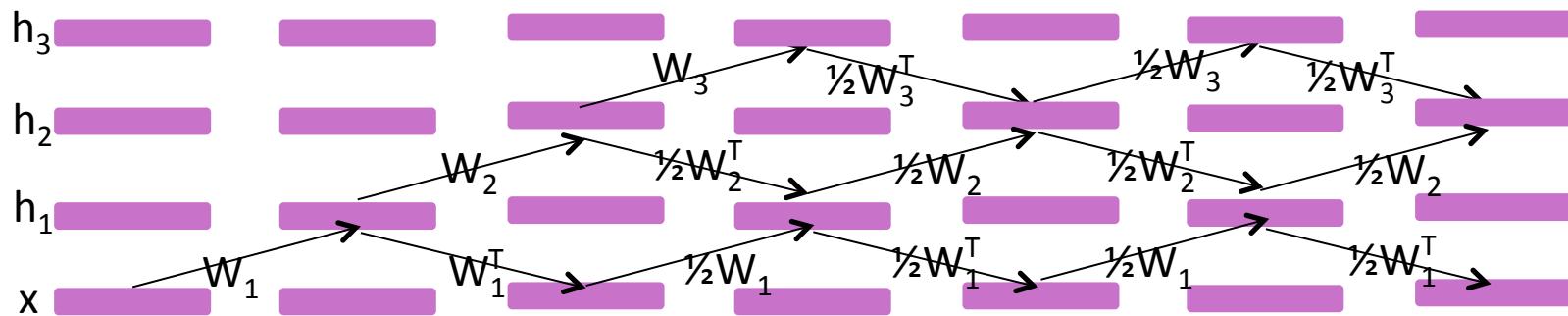


Stack of RBMs → Deep Boltzmann Machine

(Salakhutdinov & Hinton AISTATS 2009)



- Halve the RBM weights because each layer now has inputs from below and from above
- Positive phase: (mean-field) variational inference = recurrent AE
- Negative phase: Gibbs sampling (stochastic units)
- train by SML/PCD

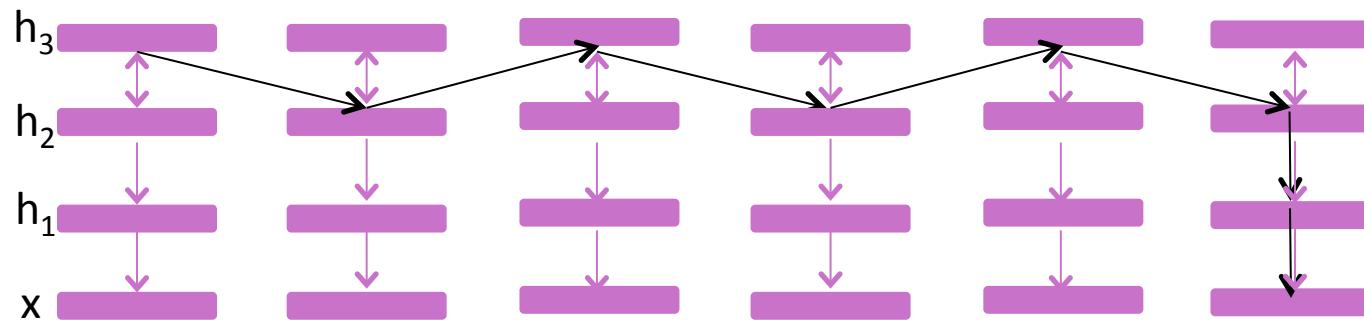


Stack of Auto-Encoders → Deep Generative Auto-Encoder

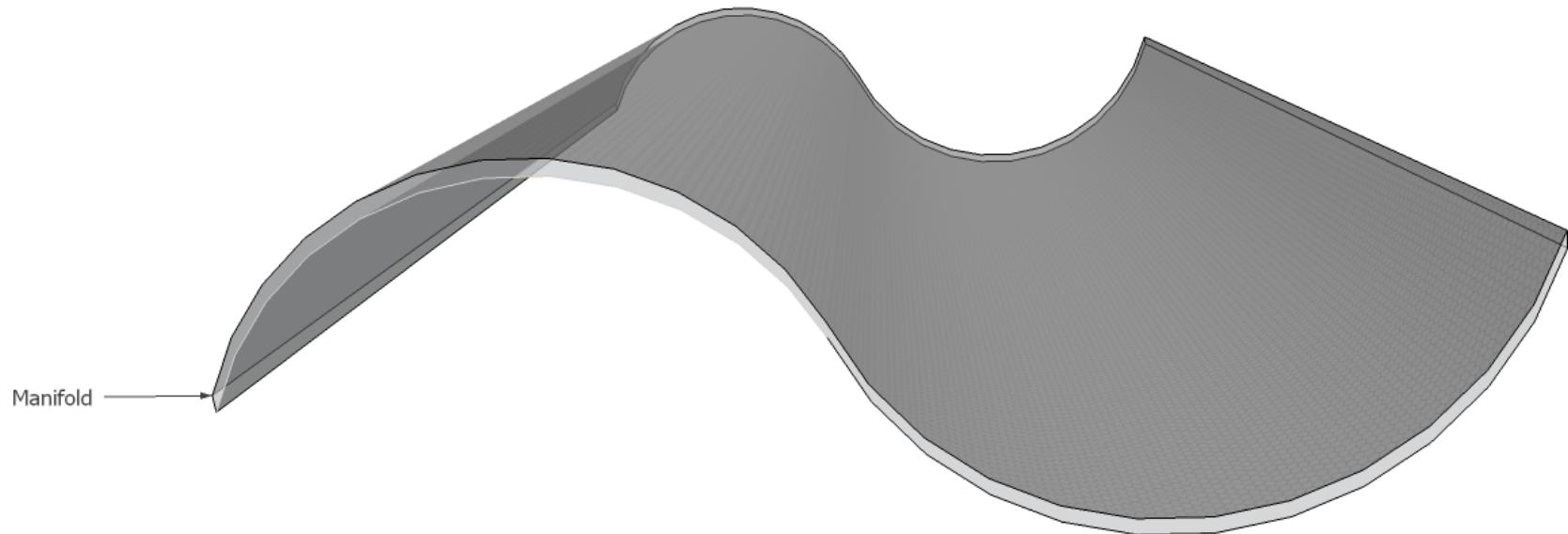
(Rifai et al ICML 2012)



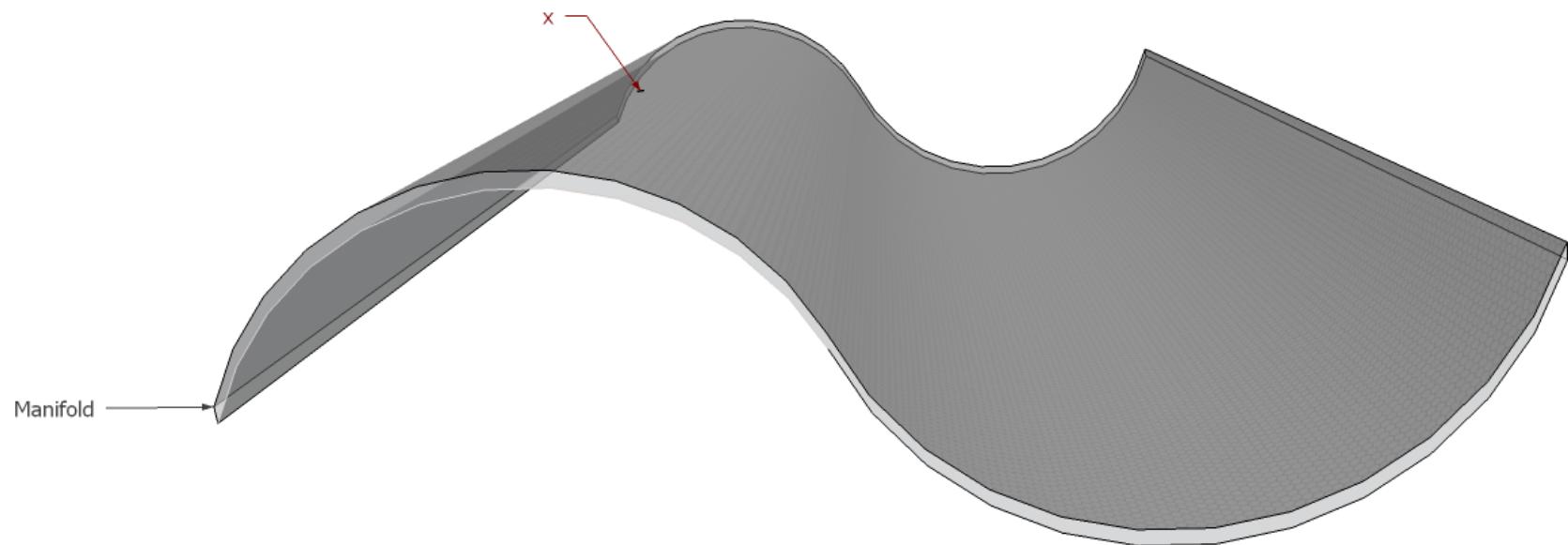
- MCMC on top-level auto-encoder
 - $h_{t+1} = \text{encode}(\text{decode}(h_t)) + \sigma \text{ noise}$
where noise is $\text{Normal}(0, d/dh \text{ encode}(\text{decode}(h_t)))$
- Then deterministically propagate down with decoders



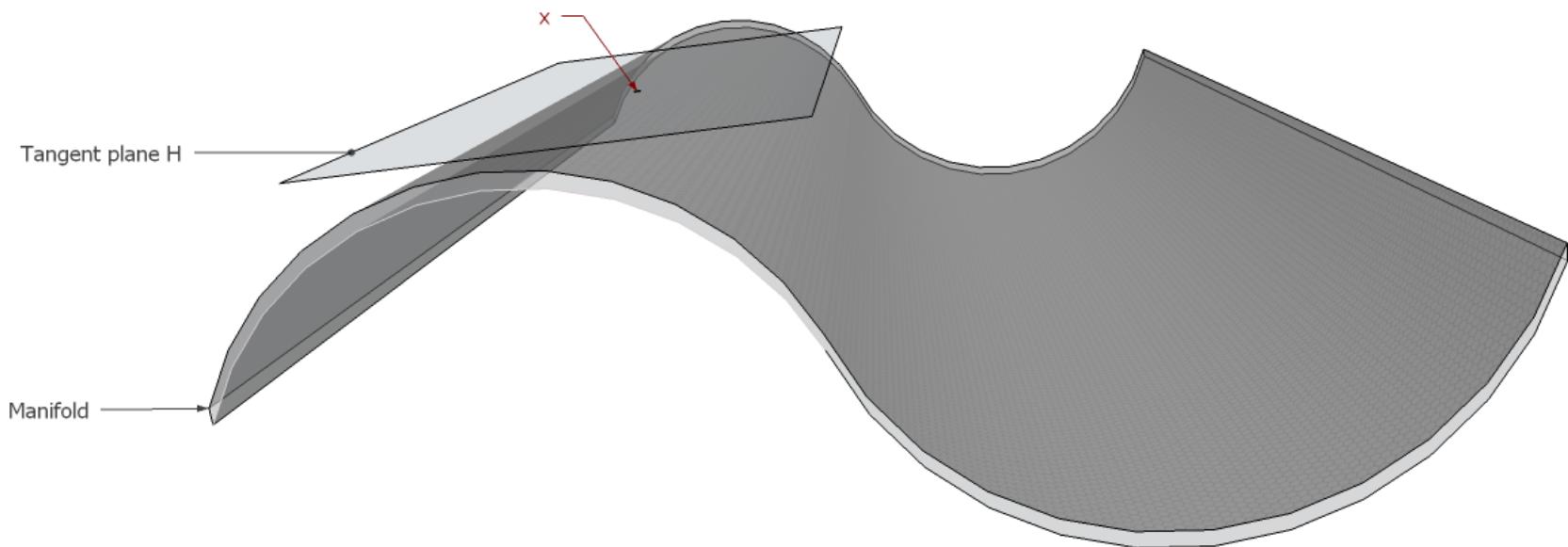
Sampling from a Regularized Auto-Encoder



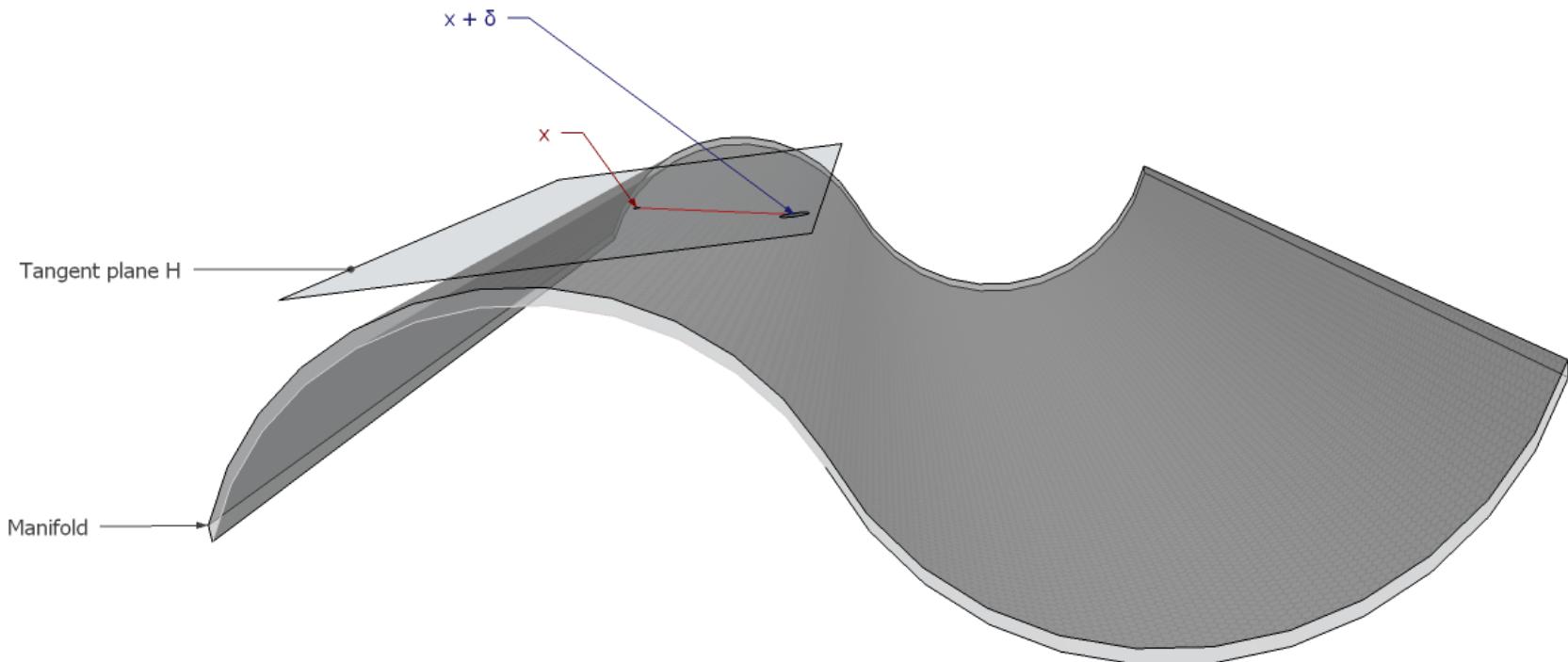
Sampling from a Regularized Auto-Encoder



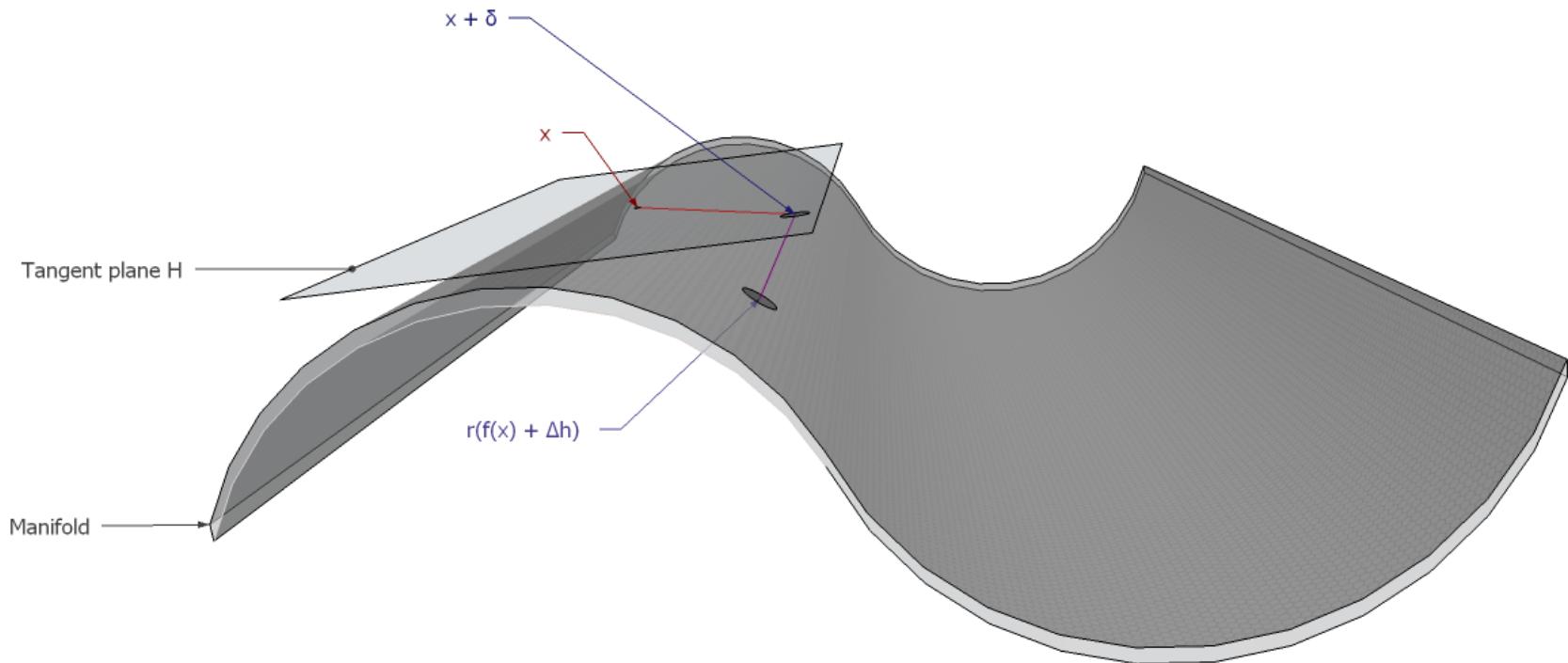
Sampling from a Regularized Auto-Encoder



Sampling from a Regularized Auto-Encoder



Sampling from a Regularized Auto-Encoder



Part 3

Practice, Issues, Questions

Deep Learning Tricks of the Trade

- Y. Bengio (2012), “Practical Recommendations for Gradient-Based Training of Deep Architectures”
 - Unsupervised pre-training
 - Stochastic gradient descent and setting learning rates
 - Main hyper-parameters
 - Learning rate schedule
 - Early stopping
 - Minibatches
 - Parameter initialization
 - Number of hidden units
 - L1 and L2 weight decay
 - Sparsity regularization
 - Debugging
 - How to efficiently search for hyper-parameter configurations



Stochastic Gradient Descent (SGD)

- Gradient descent uses total gradient over all examples per update, SGD updates after only 1 or few examples:

$$\theta^{(t)} \leftarrow \theta^{(t-1)} - \epsilon_t \frac{\partial L(z_t, \theta)}{\partial \theta}$$

- L = loss function, z_t = current example, θ = parameter vector, and ϵ_t = learning rate.
- Ordinary gradient descent is a batch method, very slow, **should never be used**. 2nd order batch method are being explored as an alternative but SGD with selected learning schedule remains the method to beat.

Learning Rates

- Simplest recipe: keep it fixed and use the same for all parameters.
- Collobert scales them by the inverse of square root of the fan-in of each neuron
- Better results can generally be obtained by allowing learning rates to decrease, typically in $O(1/t)$ because of theoretical convergence guarantees, e.g.,

$$\epsilon_t = \frac{\epsilon_0 \tau}{\max(t, \tau)}$$

with hyper-parameters ϵ_0 and τ .

Long-Term Dependencies and Clipping Trick



- In very deep networks such as recurrent networks (or possibly recursive ones), the gradient is a product of Jacobian matrices, each associated with a step in the forward computation. This can become very small or very large quickly [Bengio et al 1994], and the locality assumption of gradient descent breaks down.

$$L = L(s_T(s_{T-1}(\dots s_{t+1}(s_t, \dots))))$$

$$\frac{\partial L}{\partial s_t} = \frac{\partial L}{\partial s_T} \frac{\partial s_T}{\partial s_{T-1}} \dots \frac{\partial s_{t+1}}{\partial s_t}$$



- The solution first introduced by Mikolov is to clip gradients to a maximum value. Makes a big difference in Recurrent Nets

Early Stopping

- Beautiful **FREE LUNCH** (no need to launch many different training runs for each value of hyper-parameter for #iterations)
- Monitor validation error during training (after visiting # examples a multiple of validation set size)
- Keep track of parameters with best validation error and report them at the end
- If error does not improve enough (with some patience), stop.

Parameter Initialization

- Initialize hidden layer biases to 0 and output (or reconstruction) biases to optimal value if weights were 0 (e.g. mean target or inverse sigmoid of mean target).
- Initialize weights $\sim \text{Uniform}(-r, r)$, r inversely proportional to fan-in (previous layer size) and fan-out (next layer size):

$$\sqrt{6 / (\text{fan-in} + \text{fan-out})}$$

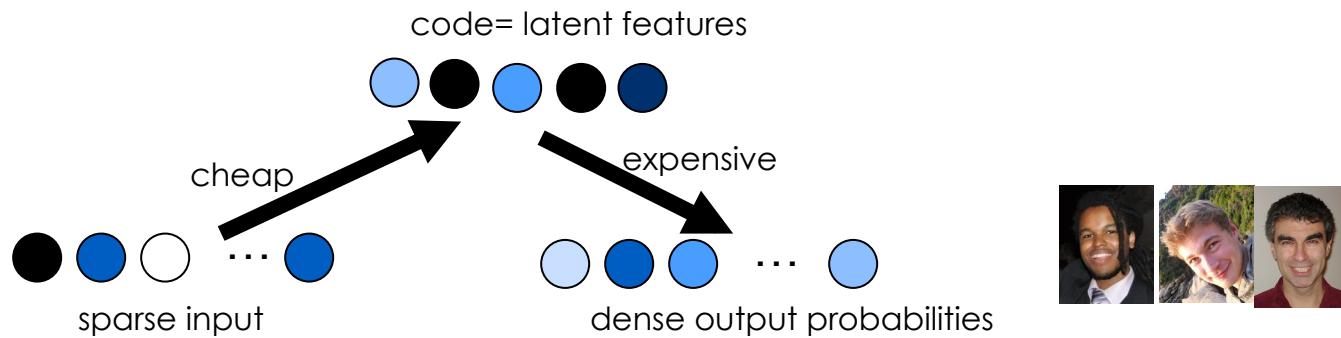
for tanh units (and 4x bigger for sigmoid units)

(Glorot & Bengio AISTATS 2010)



Handling Large Output Spaces

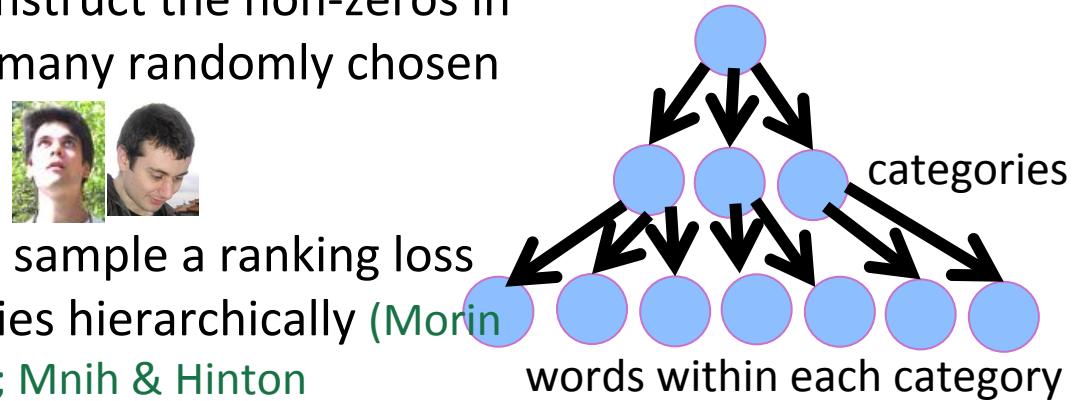
- Auto-encoders and RBMs reconstruct the input, which is sparse and high-dimensional; Language models have huge output space.



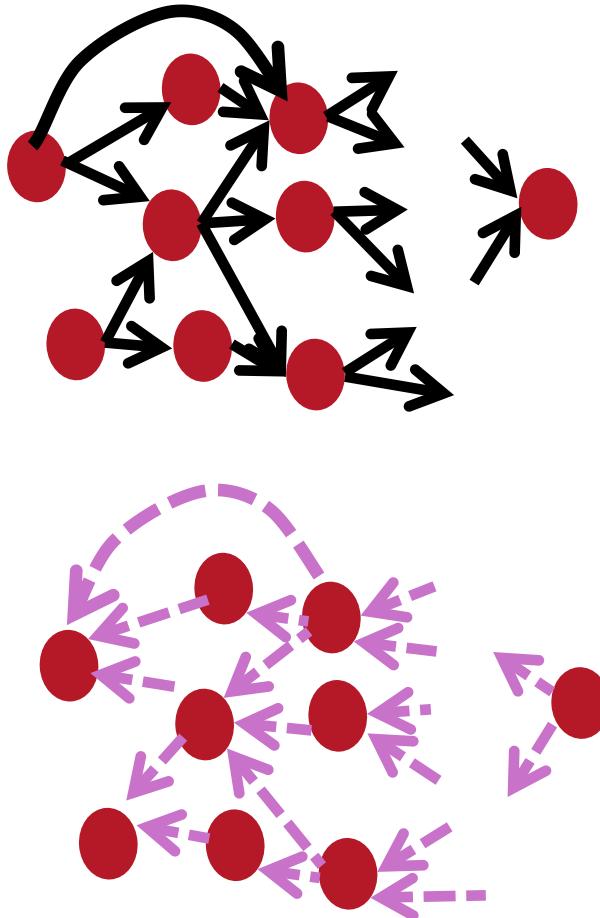
- (Dauphin et al, ICML 2011) Reconstruct the non-zeros in the input, and reconstruct as many randomly chosen zeros, + importance weights



- (Collobert & Weston, ICML 2008) sample a ranking loss
- Decompose output probabilities hierarchically (Morin & Bengio 2005; Blitzer et al 2005; Mnih & Hinton 2007,2009; Mikolov et al 2011)



Automatic Differentiation



- The gradient computation can be automatically inferred from the symbolic expression of the fprop.
- Makes it easier to quickly and safely try new models.
- 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.
- Theano Library (python) does it symbolically. Other neural network packages (Torch, Lush) can compute gradients for any given run-time value.

(Bergstra et al SciPy'2010)



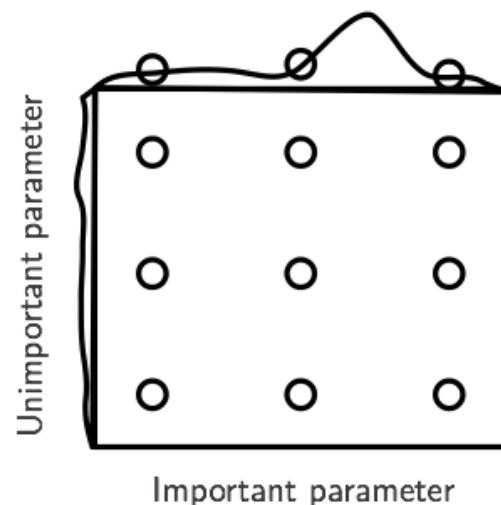
Random Sampling of Hyperparameters

(Bergstra & Bengio 2012)

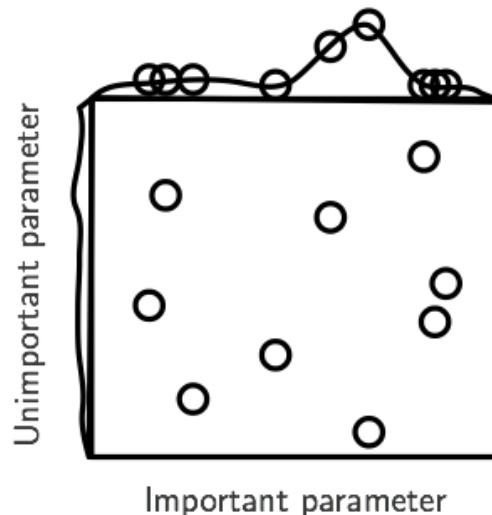


- Common approach: manual + grid search
- Grid search over hyperparameters: simple & wasteful
- Random search: simple & efficient
 - Independently sample each HP, e.g. $l.rate \sim \exp(U[\log(.1), \log(.0001)])$
 - Each training trial is iid
 - If a HP is irrelevant grid search is wasteful
 - More convenient: ok to early-stop, continue further, etc.

Grid Layout



Random Layout



Issues and Questions

Why is Unsupervised Pre-Training Working So Well?

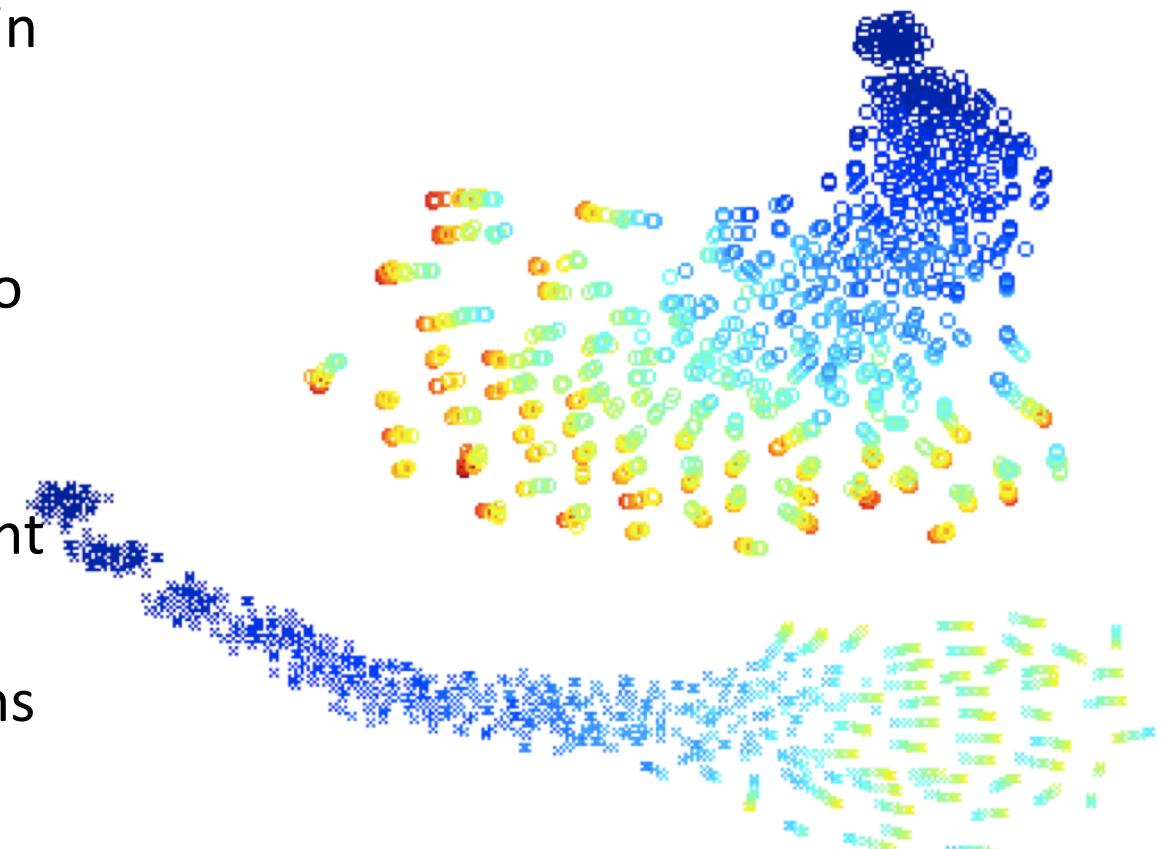
- Regularization hypothesis:
 - Unsupervised component forces model close to $P(x)$
 - Representations good for $P(x)$ are good for $P(y|x)$
- Optimization hypothesis:
 - Unsupervised initialization near better local minimum of $P(y|x)$
 - Can reach lower local minimum otherwise not achievable by random initialization
 - Easier to train each layer using a layer-local criterion

(Erhan et al JMLR 2010)



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



Dealing with a Partition Function

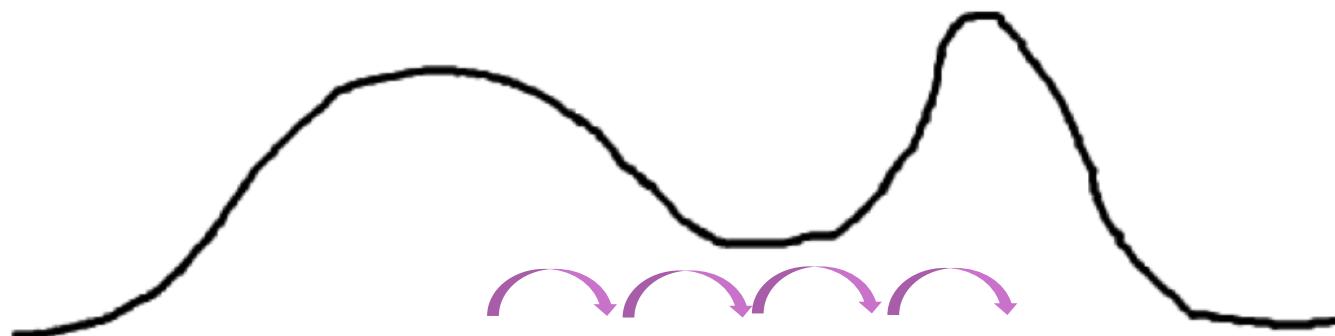
- $Z = \sum_{x,h} e^{-\text{energy}(x,h)}$
- Intractable for most interesting models
- MCMC estimators of its gradient
- Noisy gradient, can't reliably cover (spurious) modes
- Alternatives:
 - Score matching (Hyvarinen 2005)
 - Noise-contrastive estimation (Gutmann & Hyvarinen 2010)
 - Pseudo-likelihood
 - Ranking criteria (wsabie) to sample negative examples (Weston et al. 2010)
 - Auto-encoders?

Dealing with Inference

- $P(h|x)$ in general intractable (e.g. non-RBM Boltzmann machine)
- But explaining away is nice
- Approximations
 - Variational approximations, e.g. see Goodfellow et al ICML 2012
(assume a unimodal posterior)
 - MCMC, but certainly not to convergence
- We would like a model where approximate inference is going to be a good approximation
 - Predictive Sparse Decomposition does that
 - Learning approx. sparse decoding (Gregor & LeCun ICML'2010)
 - Estimating $E[h|x]$ in a Boltzmann with a separate network (Salakhutdinov & Larochelle AISTATS 2010)

For gradient & inference:
More difficult to mix with better
trained models

- Early during training, density smeared out, mode bumps overlap



- Later on, hard to cross empty voids between modes



Poor Mixing: Depth to the Rescue

- Deeper representations can yield some disentangling
- Hypotheses:
 - more abstract/disentangled representation unfold manifolds and fill more the space
 - can be exploited for better mixing between modes
 - E.g. reverse video bit, class bits in learned object representations: easy to Gibbs sample between modes at abstract level

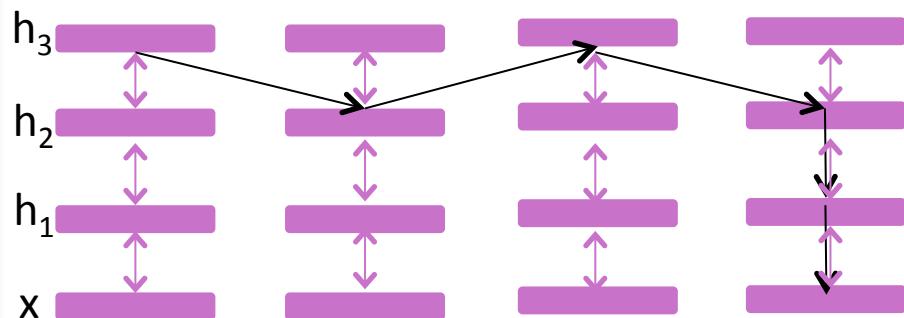
Layer abstract level

0	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
1	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
2	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3

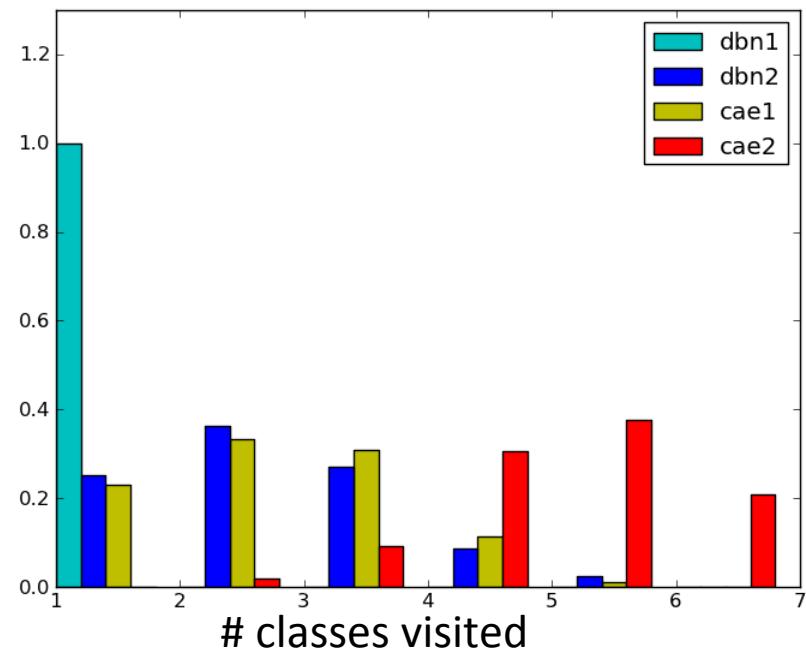
Points on the interpolating line between two classes, at different levels of representation

Poor Mixing: Depth to the Rescue

- Sampling from DBNs and stacked Contrastive Auto-Encoders:
 1. MCMC sample from top-level singler-layer model
 2. Propagate top-level representations to input-level repr.
- Visits modes (classes) faster



Toronto Face Database



What are regularized auto-encoders learning exactly?

- Any training criterion $E(X, \theta)$ interpretable as a form of MAP:
- JEPADA: Joint Energy in PArameters and Data (Bengio, Courville, Vincent 2012)

$$P(X, \theta) = \frac{e^{-E(X, \theta)}}{Z}$$

This Z does not depend on θ . If $E(X, \theta)$ tractable, so is the gradient
No magic; consider traditional directed model:

$$E(X, \theta) = E_\theta(X) + \log Z_\theta - \log P(\theta)$$

Application: Predictive Sparse Decomposition, regularized auto-encoders, ...

What are regularized auto-encoders learning exactly?

- Denoising auto-encoder is also contractive

$$\begin{aligned} E[\ell(x, r(x + \epsilon))] &\approx E \left[\left(x - \left(r(x) + \frac{\partial r(x)}{\partial x} \epsilon \right) \right)^T \left(x - \left(r(x) + \frac{\partial r(x)}{\partial x} \epsilon \right) \right) \right] \\ &= E[\|x - r(x)\|^2] + \sigma^2 E \left[\left\| \frac{\partial r(x)}{\partial x} \right\|_F^2 \right] \end{aligned}$$

- Contractive/denoising auto-encoders learn **local moments**
 - $r(x)-x$ estimates the direction of $E[X|X \text{ in ball around } x]$
 - Jacobian $\frac{\partial r(x)}{\partial x}$ estimates $\text{Cov}(X|X \text{ in ball around } x)$
- These two also respectively estimate the score and (roughly) the Hessian of the density

More Open Questions

- What is a good representation? Disentangling factors? Can we design better training criteria / setups?
- Can we safely assume $P(h|x)$ to be unimodal or few-modal? If not, is there any alternative to explicit latent variables?
- Should we have explicit explaining away or just learn to produce good representations?
- Should learned representations be low-dimensional or sparse/ saturated and high-dimensional?
- Why is it more difficult to optimize deeper (or recurrent/ recursive) architectures? Does it necessarily get more difficult as training progresses? Can we do better?

The End

