

# Deep Learning

Universita di Siena  
Italy

Yoshua Bengio

June 30 – July 4, 2014



# Outline of the Tutorial

1. Representation Learning, motivations
2. Algorithms
  - Feedforward deep networks
  - Convolutional nets
  - Recurrent and recursive nets
  - Regularization and optimization
  - RBMs, DBMs and DBNs
  - Regularized auto-encoders
3. Practical Considerations and Applications
4. Challenges & Ongoing Work

Upcoming book: “Deep Learning” <http://www.iro.umontreal.ca/~bengioy/dlbook/> for a pdf of the slides and draft chapters of the book.



# Ultimate Goal

- **Understand the principles giving rise to intelligence**

# Focus

- **Learning:** mathematical and computational principles allowing one to learn from examples in order to acquire knowledge

# Breakthrough

- **Deep Learning:** machine learning algorithms inspired by brains, based on learning multiple levels of representation / abstraction.

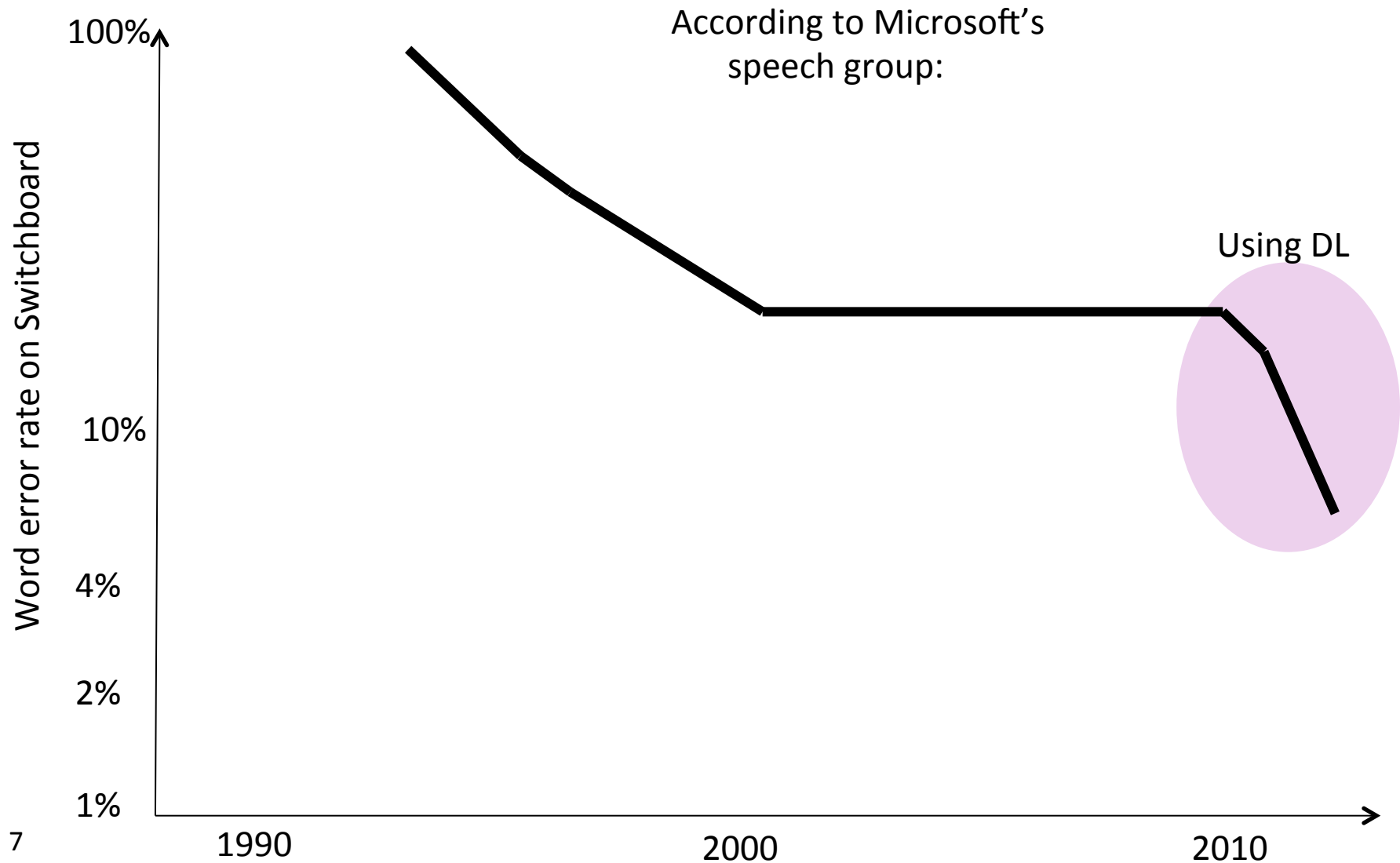
# Economic Impact

**Deep learning has revolutionized**

- **Speech recognition**
- **Object recognition**

**More to come,** including other areas of computer vision, NLP, dialogue...

# The dramatic impact of Deep Learning on Speech Recognition



# Object Recognition Breakthrough



## ImageNet Breakthrough

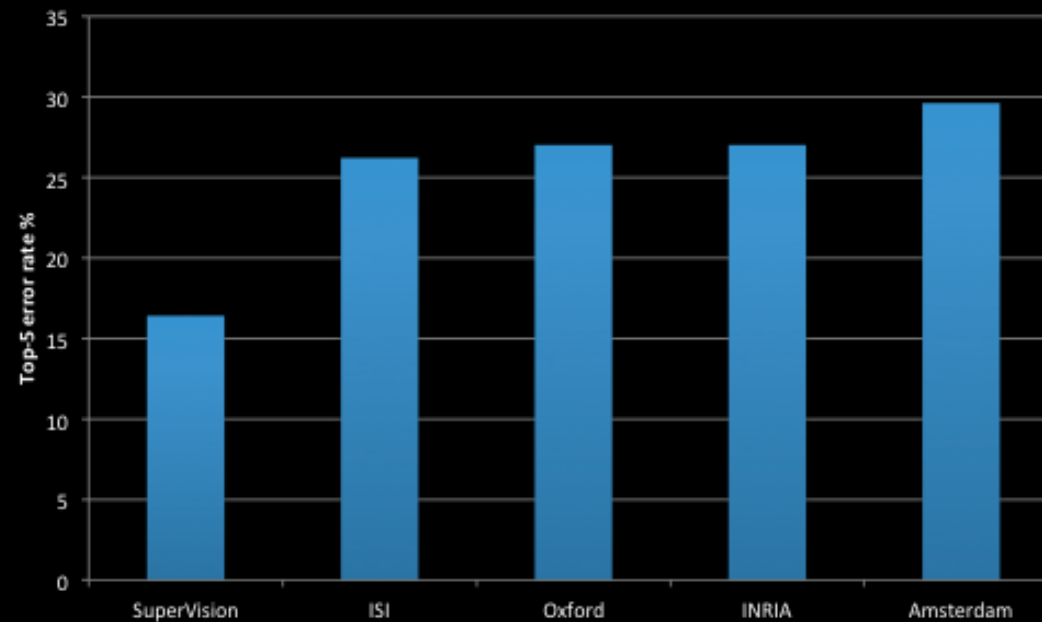
- Achieves state-of-the-art on many object recognition tasks.

See: [deeplearning.cs.toronto.edu](http://deeplearning.cs.toronto.edu)

# ImageNet Classification 2012

---

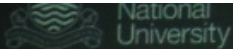
- Krizhevsky et al. -- 16.4% error (top-5)
- Next best (non-convnet) – 26.2% error



Slide from Rob Fergus, NIPS tutorial, 2012

# Montreal Deep Nets Win Emotion Recognition in the Wild Challenge

Predict emotional expression from video (using images + audio)



**Results!**

Team Name	Classification accuracy	
Audio baseline	22.4 %	
Video baseline	22.7 %	
Fusion	27.5 %	
Nottingham	24.7 %	
Oulu	21.5 %	
KIT	29.8 %	
UCSD	37.1 %	2nd
ICT@CAS	35.9 %	3rd
York	27.6 %	
LNMIIT	20.5 %	
Montreal	41.0 %	1st
Ulm	27.2 %	

Dec. 9, 2013



# 10 BREAKTHROUGH TECHNOLOGIES 2013

Intr

## Deep Learning

With massive amounts of computational power, machines can now recognize objects and translate speech in real time. Artificial intelligence is finally getting smart. →

## Temporary Social Media

Messages that quickly self-destruct could enhance the privacy of online communications and make people freer to be spontaneous. →

## Prenatal DNA Sequencing

Reading the DNA of fetuses will be the next frontier of the genomic revolution. But do you really want to know about the genetic problems or musical aptitude of your unborn child? →

## Adv Man

Ske  
prin  
wor  
mar  
the  
tech  
jet p

## Memory Implants

A maverick neuroscientist believes he has deciphered the code by which the brain

## Smart Watches

## Ultra-Efficient Solar Power

Doubling the efficiency of a solar cell would completely

## Big Pho

Coll  
ana  
from  
pho

# Deep Learning in the News



EXCLUSIVE

## Facebook, Google in 'Deep Learning' Arms Race

Yann LeCun, an NYU artificial intelligence researcher who now works for Facebook. Photo: Josh Valcarcel/WIRED



NEWS BULLETIN

## Google Beat Facebook for DeepMind

### Google Acquires Artificial Intelligence Startup DeepMind For More Than \$500M

Posted Jan 26, 2014 by [Catherine Shu](#) (@catherineshu)

# Some Applications of DL

- **Language Modeling** (Speech Recognition, Machine Translation)
- **Acoustic Modeling** (**speech recognition**, music modeling)
- **NLP syntactic/semantic tagging** (Part-Of-Speech, chunking, Named Entity Recognition, Semantic Role Labeling, Parsing)
- **NLP applications**: sentiment analysis, paraphrasing, question-answering, Word-Sense Disambiguation
- **Object recognition in images**: photo search and image search: handwriting recognition, document analysis, handwriting synthesis, **superhuman traffic sign classification**, street view house numbers, **emotion detection from facial images**, roads from satellites.
- **Personalization**/recommendation/fraud/ads
- **Molecular properties**: QSAR, quantum calculations



# Challenges

- **Unsupervised Learning & Reinforcement Learning**
  - *Intractable* computations
  - Key to more adaptable models and complex output decisions
- **Scaling up to much larger models (& Big Data)**
- **Expanding the scope of deep learning applications**

# Potential Outcome: AI

- **Computers that can**
  - **see and hear**
  - **understand natural language**
  - **understand human behavior**
- **Better understanding of human & animal intelligence**
- **Personal assistants, self-driving cars...**

# Ultimate Goals

- **AI**
- Needs **knowledge**
- Needs **learning**  
(involves priors + *optimization/search*)
- Needs **generalization**  
(guessing where probability mass concentrates)
- Needs ways to fight the curse of dimensionality  
(exponentially many configurations of the variables to consider)
- Needs disentangling the underlying explanatory factors  
(making sense of the data)

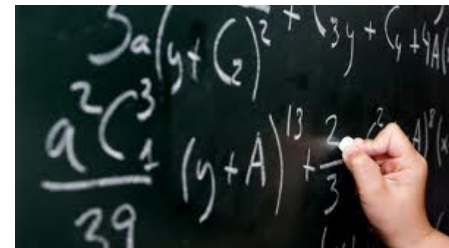
## Part 1

# Intro & Motivations for Representation Learning and Deep Learning

# What is Machine Learning?

Mathematical principles and computer algorithms exploiting data

- for extracting what is **GENERAL**

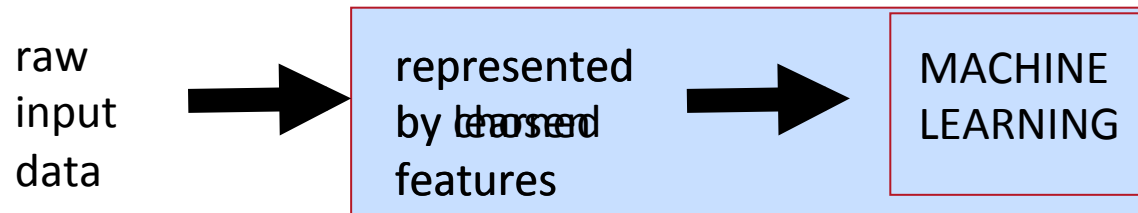


- so as to be able to say something meaningful **about new cases**
- to identify which **configurations** of variables are **plausible**
- to **generate** new plausible configurations or **choose best ones**
- to learn to **predict, classify, take decisions**

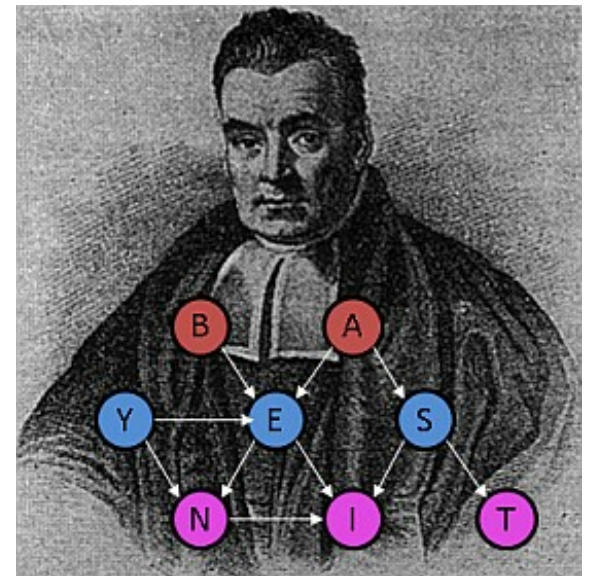


# Representation Learning

- Good **features** essential for successful ML: 90% of effort



- Handcrafting features vs learning them
- Good representation?
- **guesses**  
the features / factors / causes



# Google Image Search:

Different object types represented in the same space



Google:

S. Bengio, J.  
Weston & N.  
Usunier



(IJCAI 2011,  
NIPS'2010,  
JMLR 2010,  
MLJ 2010)



$\Phi_I(\cdot)$

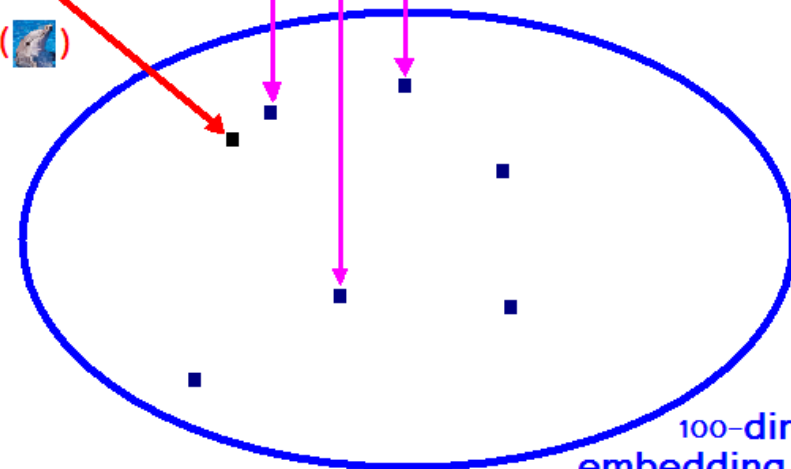
$\Phi_W(\text{DOLPHIN})$

DOLPHIN

OBAMA

EIFFEL TOWER

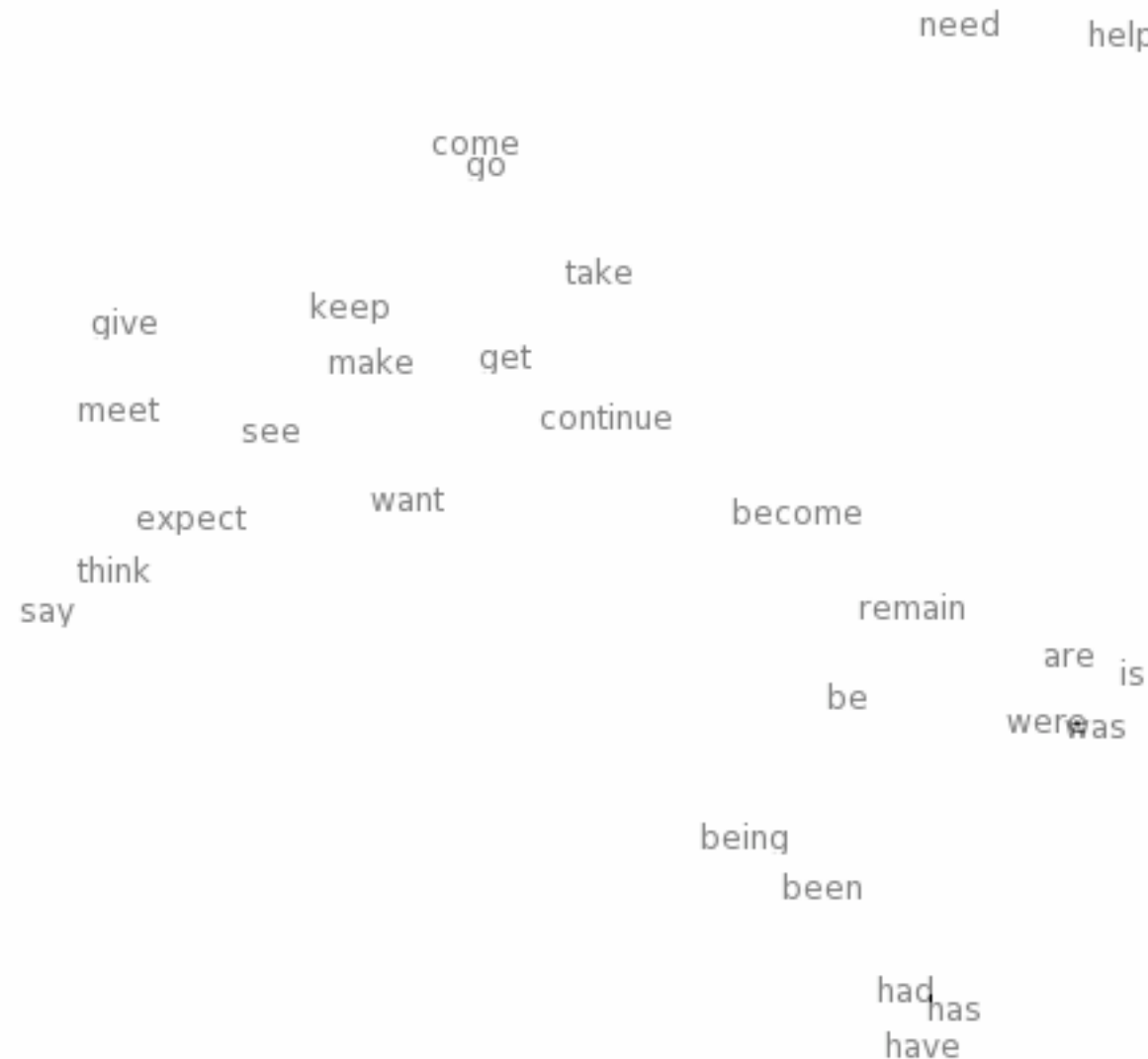
.....



100-dim  
embedding space

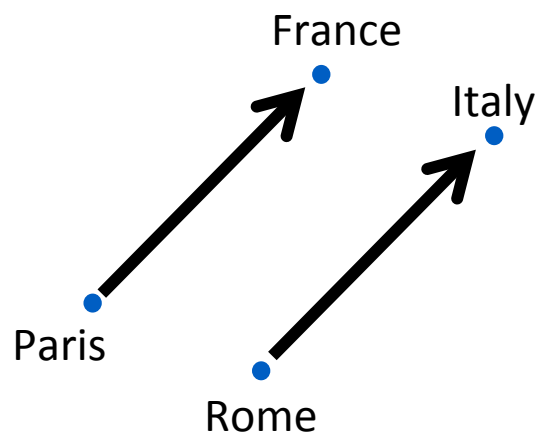
Learn  $\Phi_I(\cdot)$  and  $\Phi_W(\cdot)$  to optimize precision@k.

## Following up on (Bengio et al NIPS'2000) Neural word embeddings - visualization



# Analogical Representations for Free (Mikolov et al, ICLR 2013)

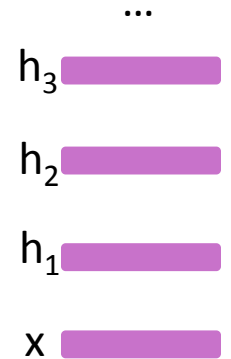
- Semantic relations appear as linear relationships in the space of learned representations
- King – Queen  $\approx$  Man – Woman
- Paris – France + Italy  $\approx$  Rome



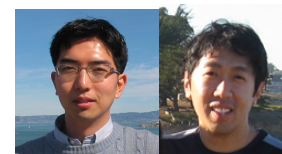
# Deep Representation Learning

Learn multiple levels of representation of increasing complexity/abstraction

- theory: exponential gain
- brains are deep
- cognition is compositional
- Better mixing (Bengio et al, ICML 2013)
- **They work! SOTA on industrial-scale AI tasks (object recognition, speech recognition, language modeling, music modeling)**



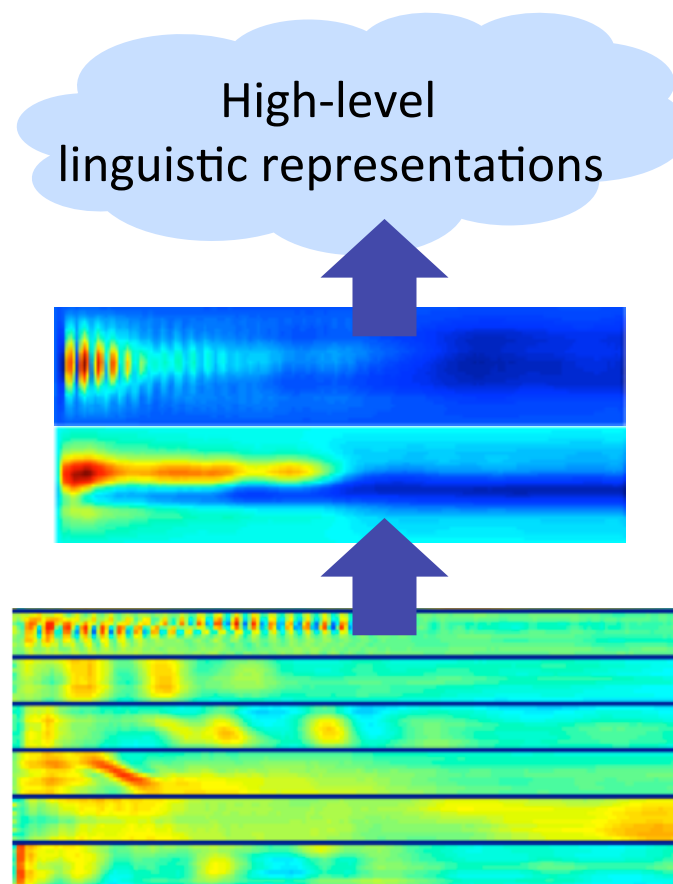
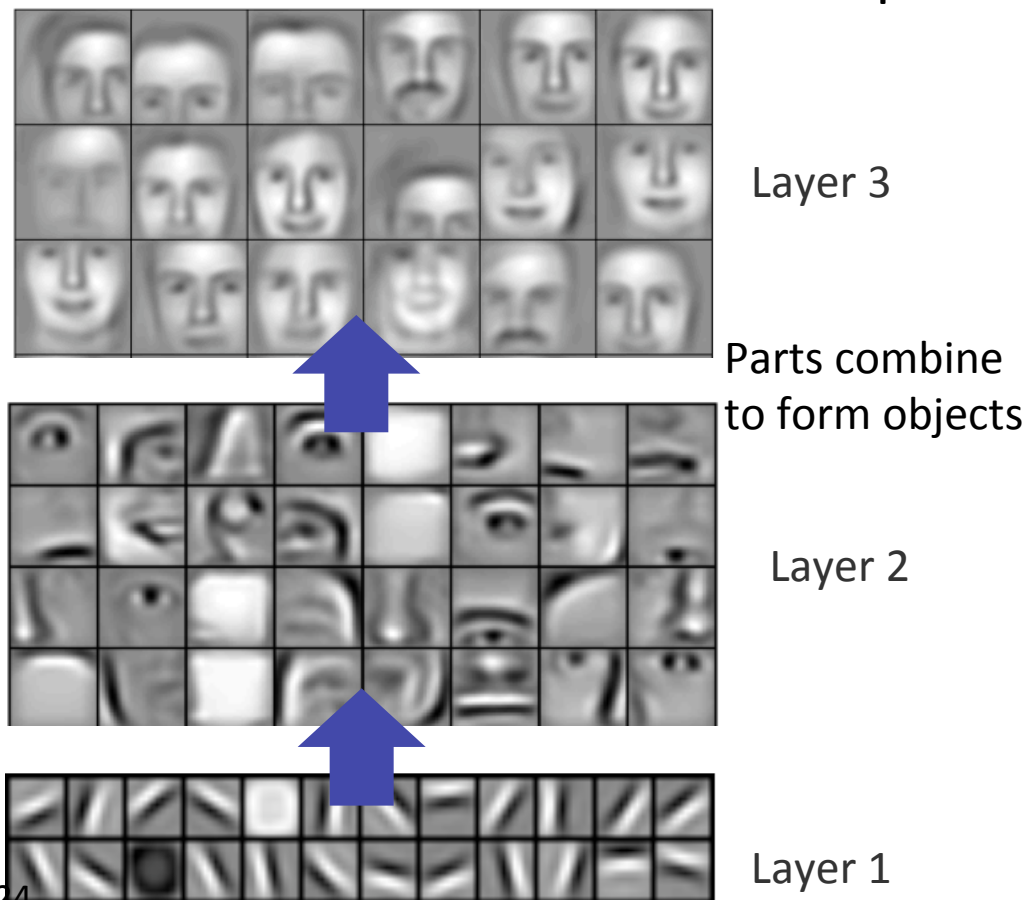
# Learning multiple levels of representation



(Lee, Largman, Pham & Ng, NIPS 2009)

(Lee, Grosse, Ranganath & Ng, ICML 2009)

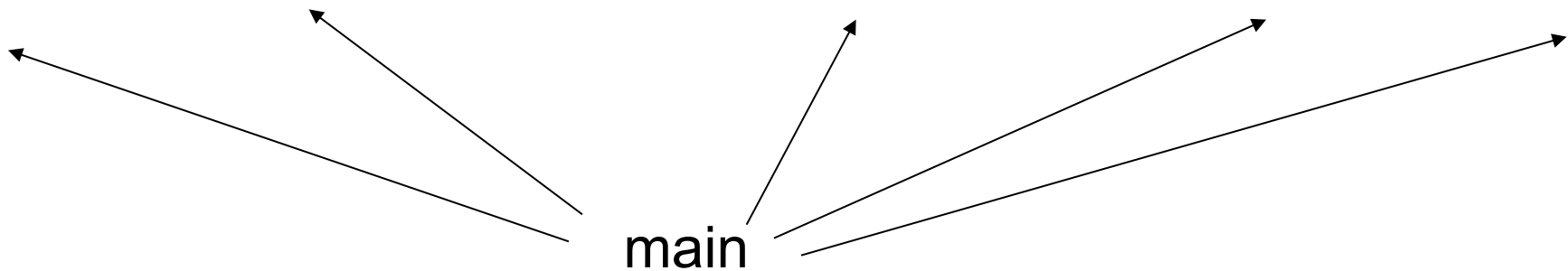
Successive model layers learn deeper intermediate representations



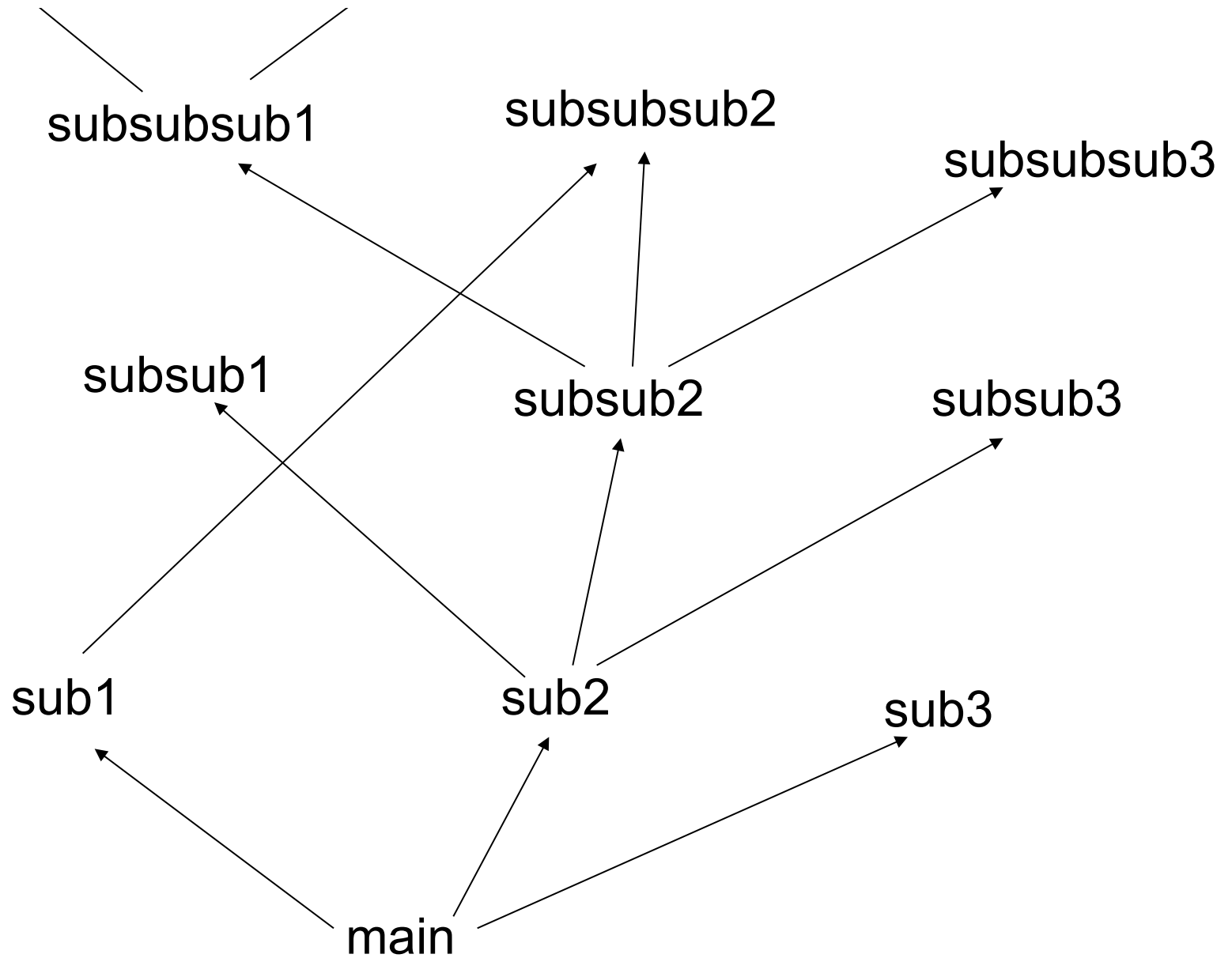
**Prior: underlying factors & concepts compactly expressed w/ multiple levels of abstraction**

subroutine1 includes  
subsub1 code and  
subsub2 code and  
subsubsub1 code

subroutine2 includes  
subsub2 code and  
subsub3 code and  
subsubsub3 code and ...



**“Shallow” computer program**



**“Deep” computer program**



# Deep Architectures are More Expressive

Theoretical arguments:

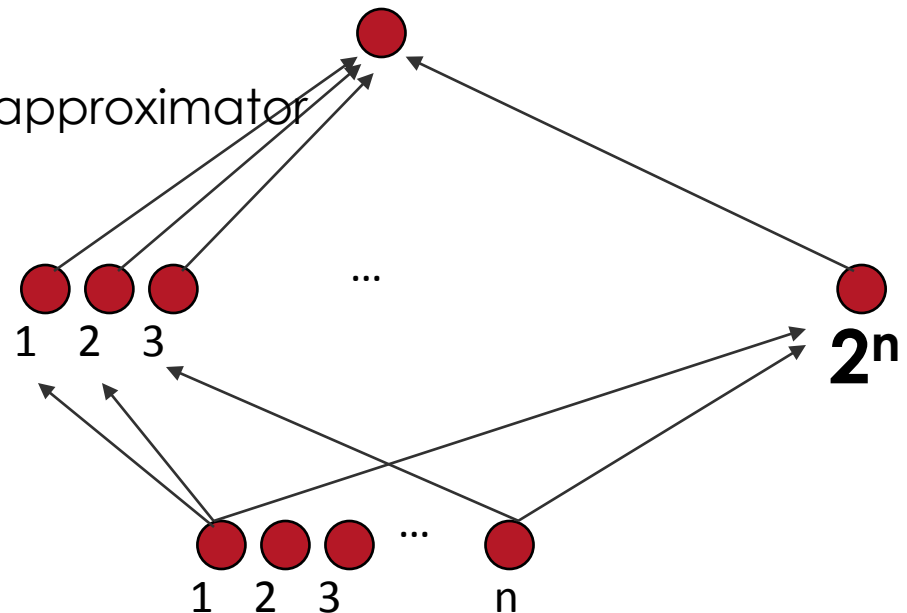
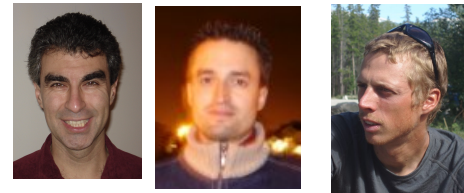
2 layers of {  
Logic gates  
Formal neurons  
RBF units

= universal approximator

RBMs & auto-encoders = universal approximator

Theorems on advantage of depth:  
(Hastad et al 86 & 91, Bengio et al 2007, Bengio & Delalleau 2011, Braverman 2011)

Some functions compactly represented with  $k$  layers may require exponential size with 2 layers



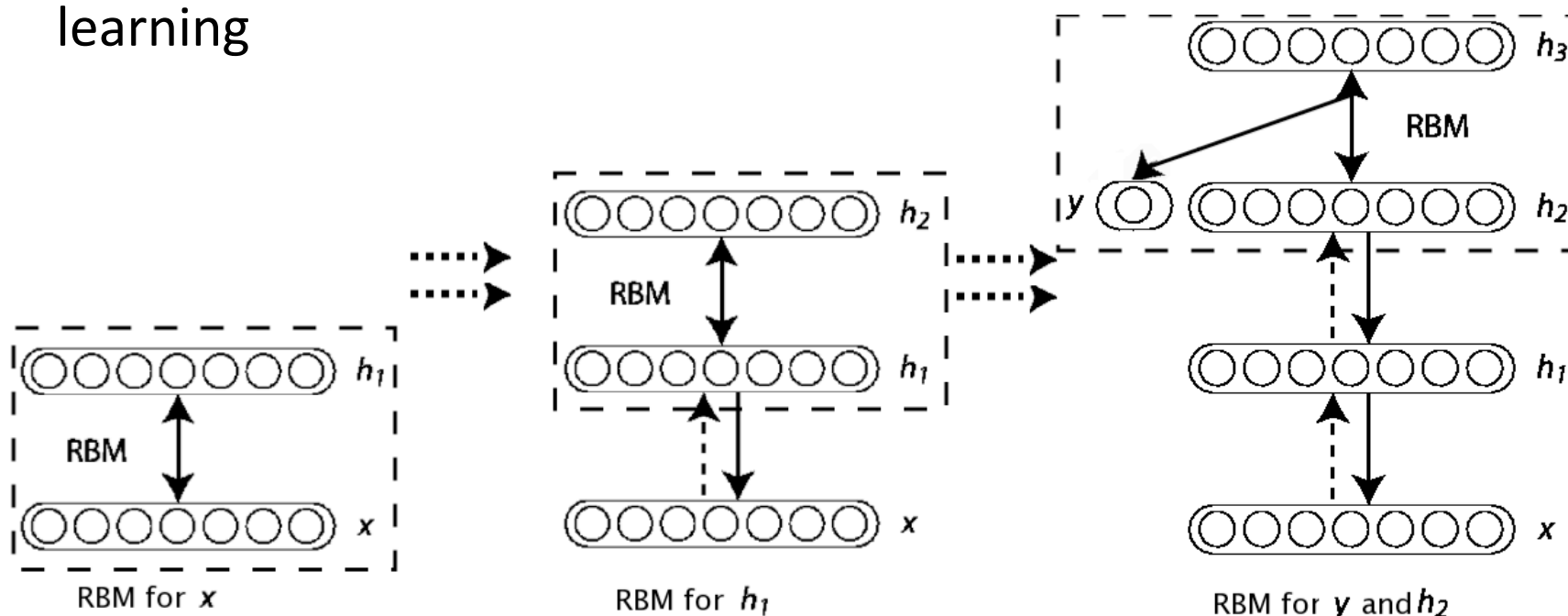
# Breakthrough in 2006

- 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



# Stacking Single-Layer Learners

- One of the big ideas from 2006: layer-wise unsupervised feature learning

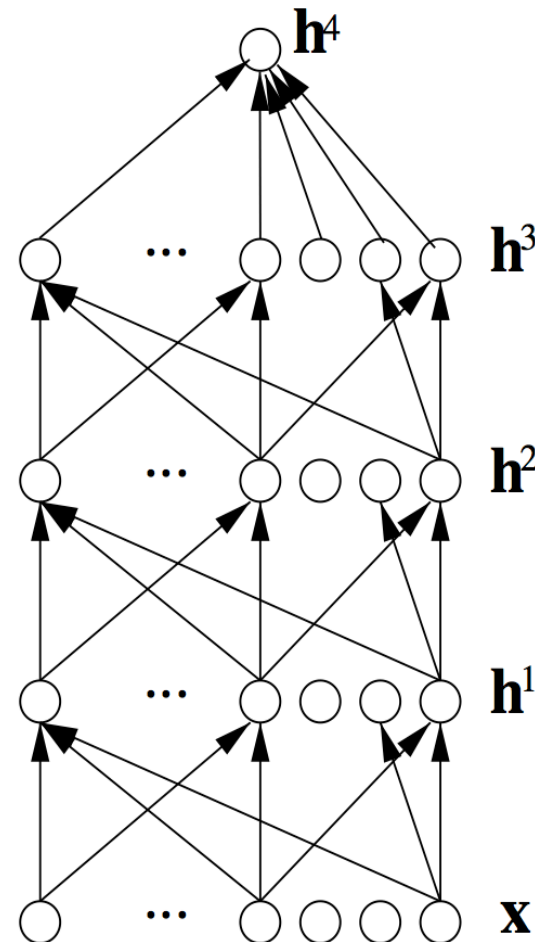


Stacking Restricted Boltzmann Machines (RBM) → Deep Belief Network (DBN)

Stacking regularized auto-encoders → deep neural nets

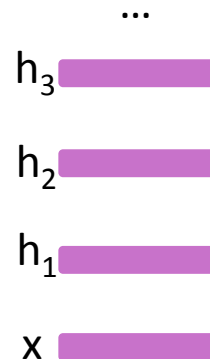
# Deep Supervised Neural Nets

- Now can train them even without unsupervised pre-training:  
**better initialization and non-linearities** (rectifiers, maxout), generalize well with large labeled sets and regularizers (dropout)
- **Unsupervised pre-training:**  
rare classes, transfer, smaller labeled sets, or as extra regularizer.

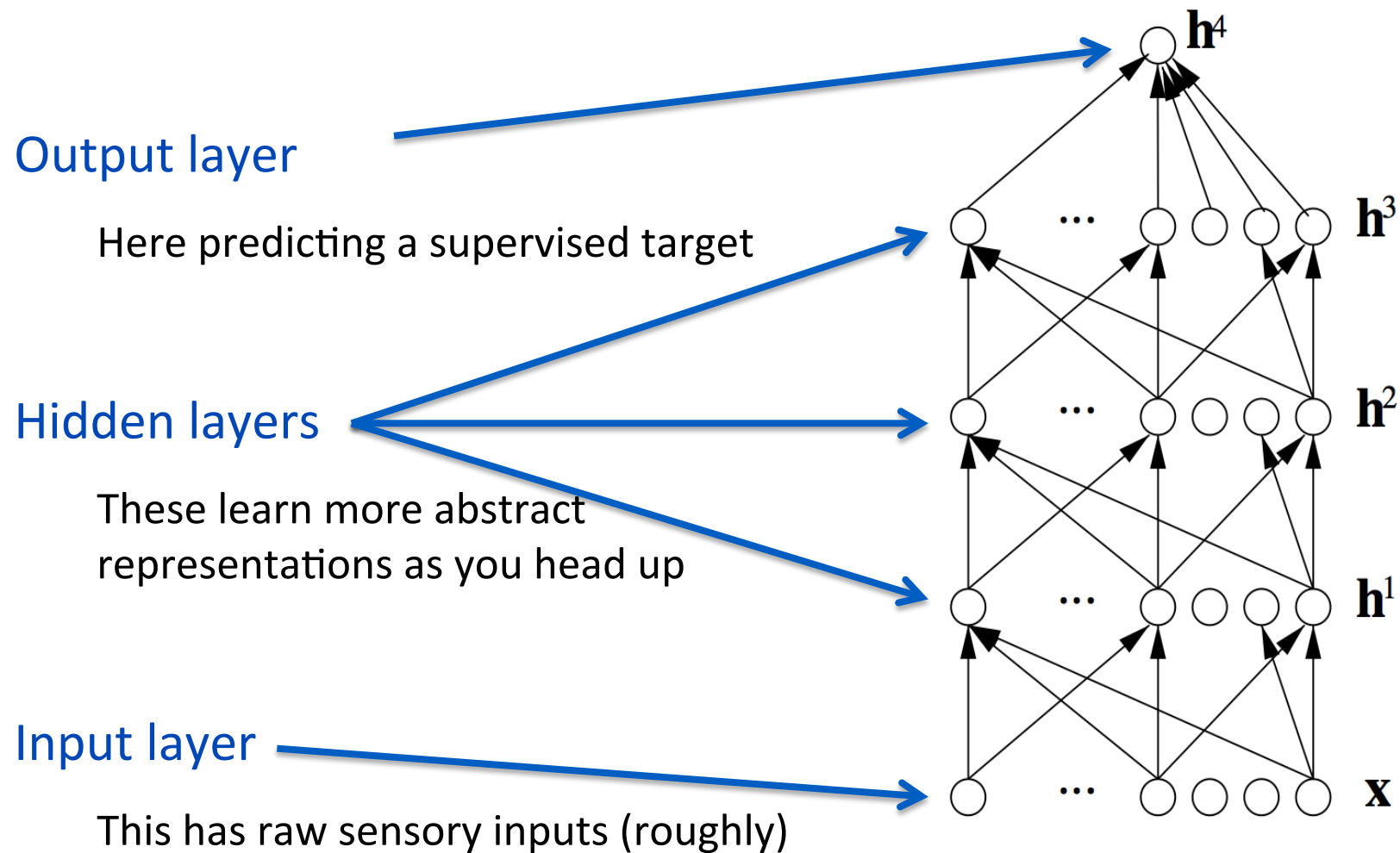


# Deep Learning

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



# A Good Old Deep Architecture: MLPs



# A (Vanilla) Modern Deep Architecture

## Optional Output layer

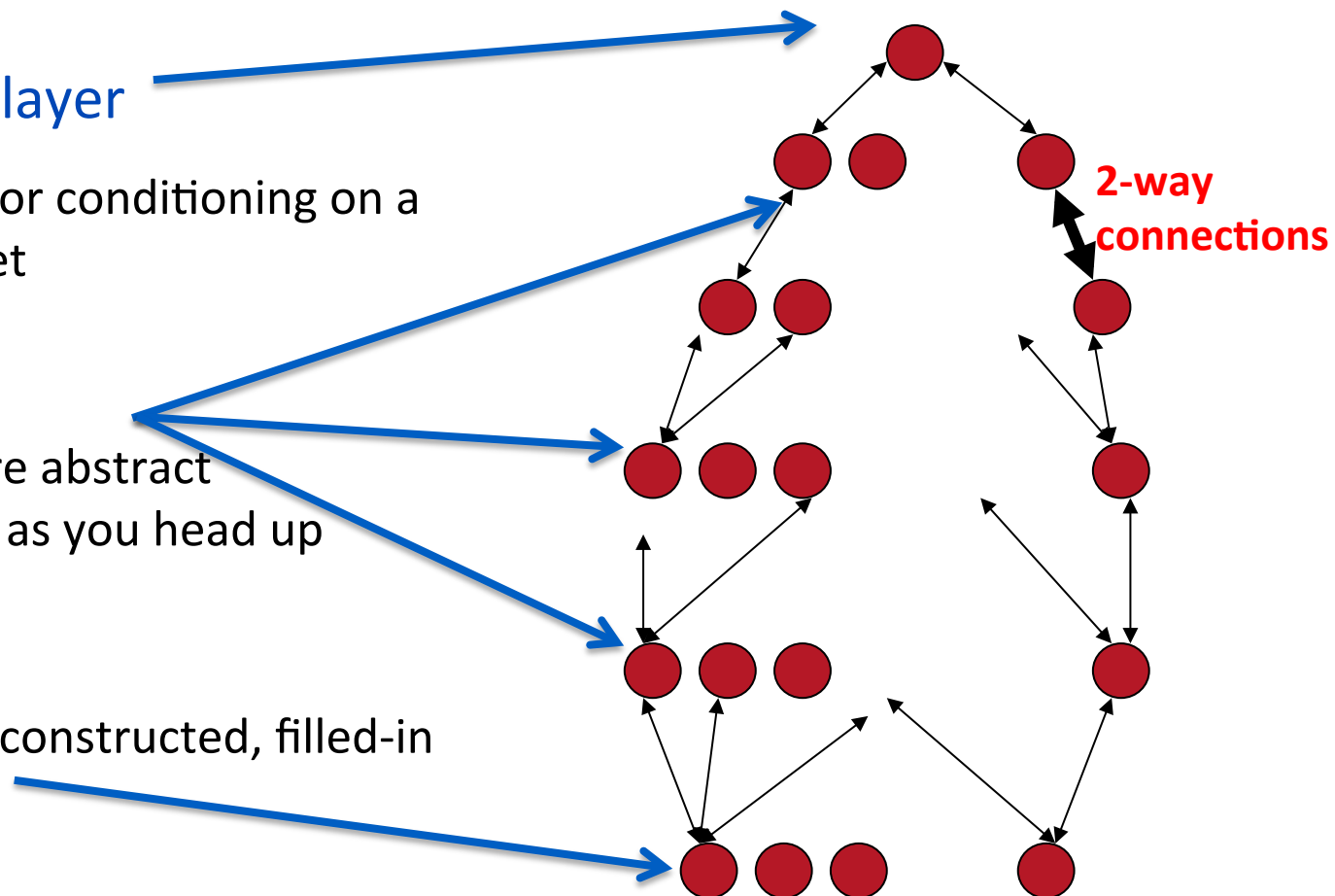
Here predicting or conditioning on a supervised target

## Hidden layers

These learn more abstract representations as you head up

## Input layer

Inputs can be reconstructed, filled-in or sampled



# What differences with Neural Nets of the 90's?

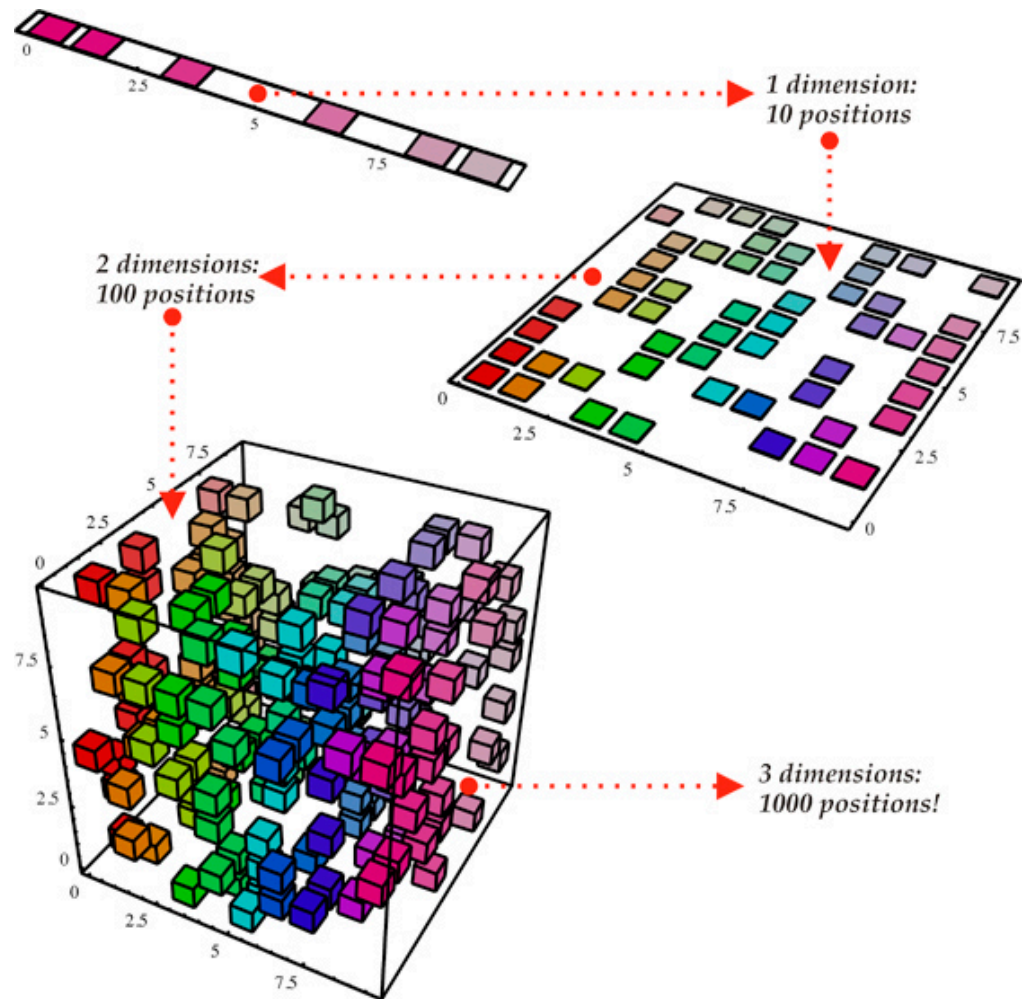
- Other kinds of hierarchies are possible (e.g. A. Yuille, D. McAllester )
- Bigger models
- Better training
  - **Initialization**: information flow (Jacobians e-values closer to 1)
  - **Symmetry breaking**: initialization, sparsity regularization and non-linearities (rectifier, maxout, etc.)
- Unsupervised and multi-task learning → better transfer learning
- Larger labeled sets: **the advantage increases!**
- Better regularizers (dropout, injected noise, temporal coherence)



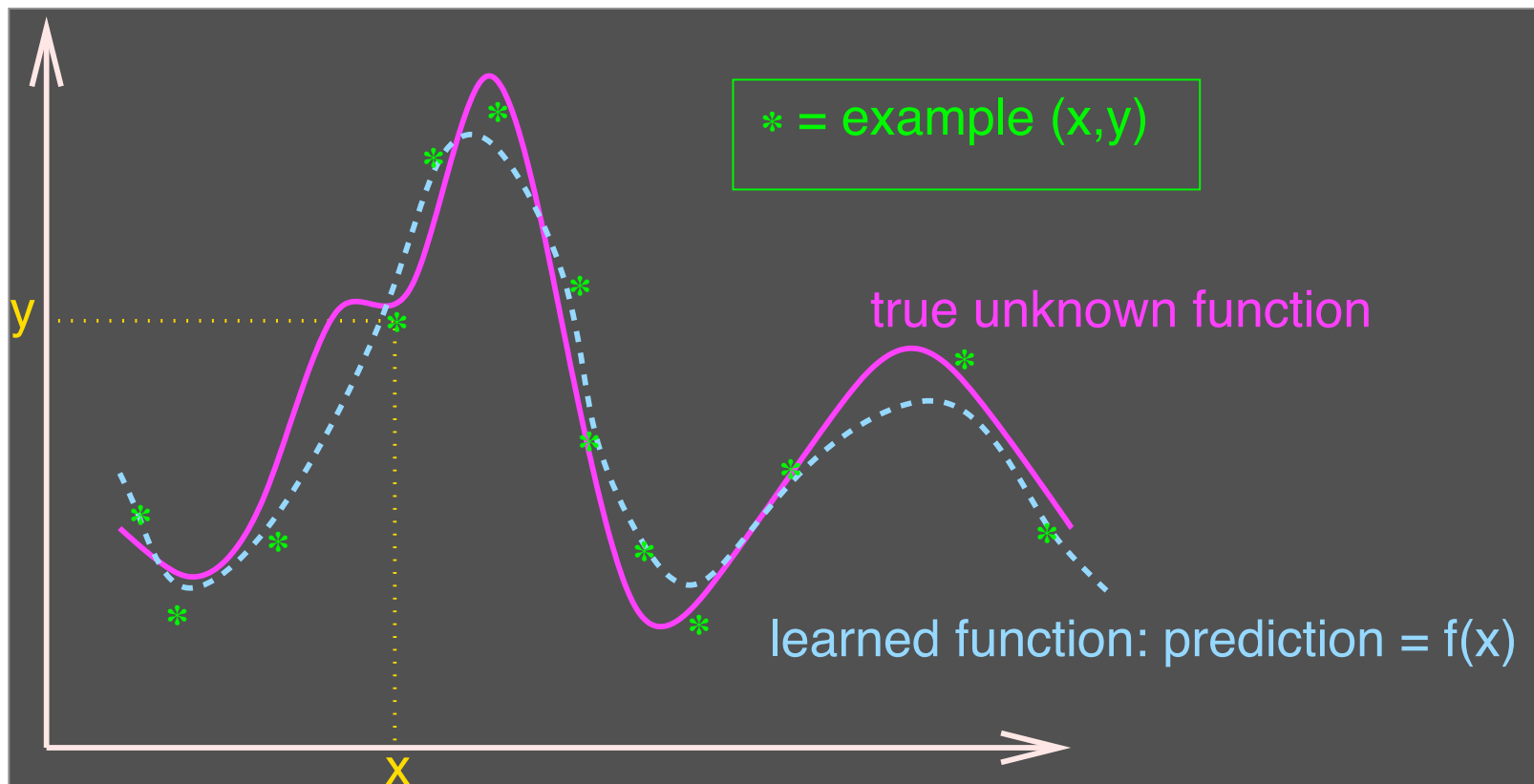
# ML 101. What We Are Fighting Against: The Curse of Dimensionality

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

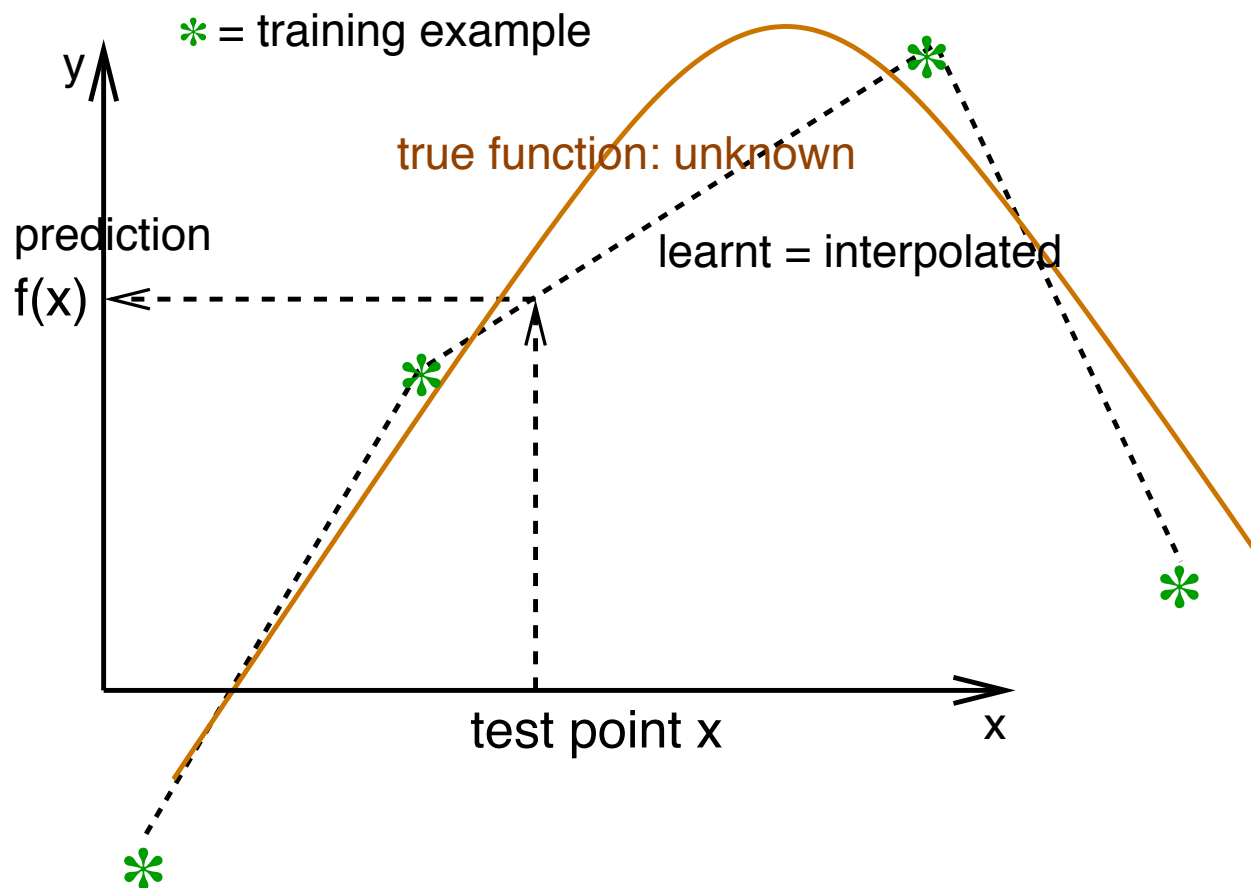
Classical solution: hope  
for a smooth enough  
target function, or  
make it smooth by  
handcrafting good  
features / kernel



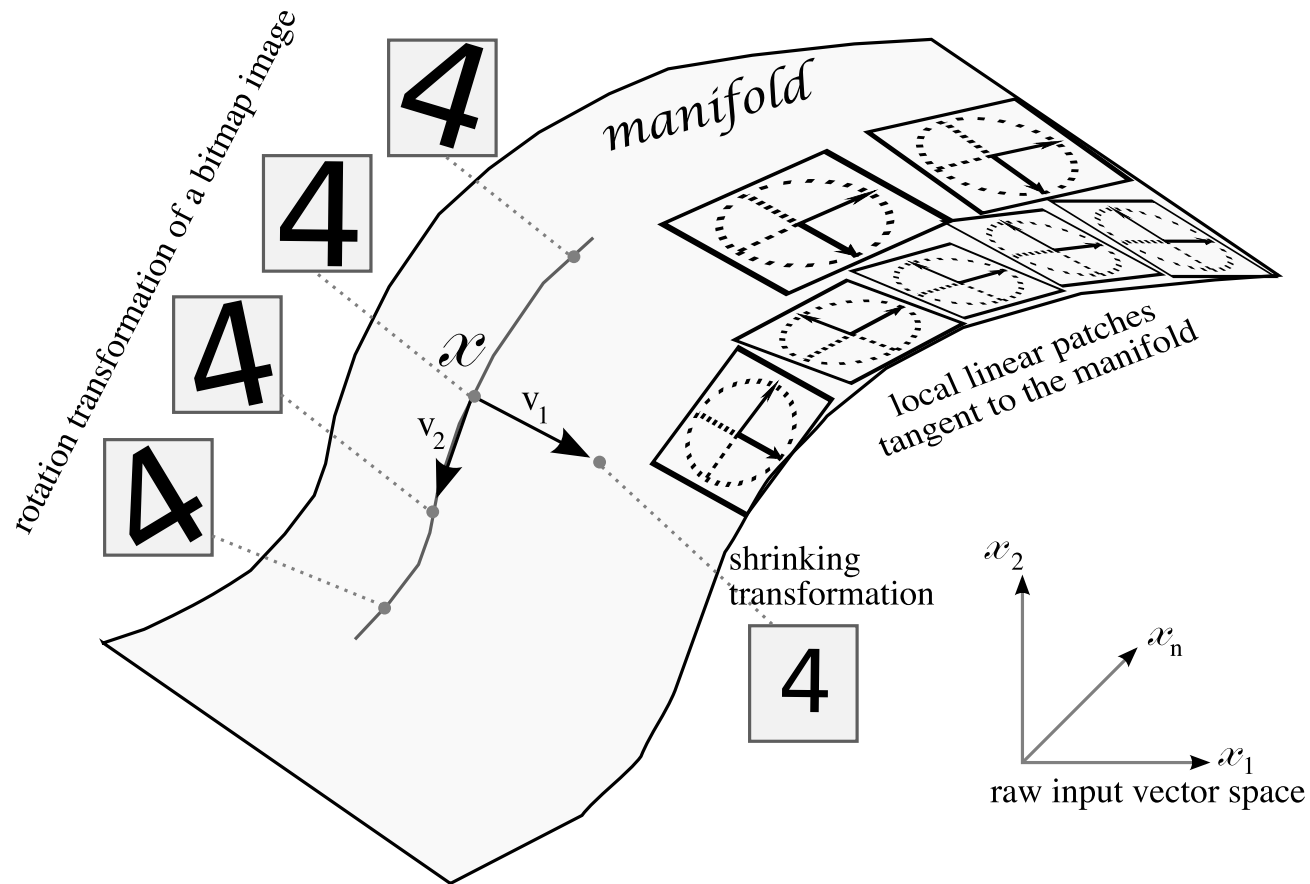
# Easy Learning



# Local Smoothness Prior: Locally Capture the Variations



## However, Real Data Are near Highly Curved Sub-Manifolds

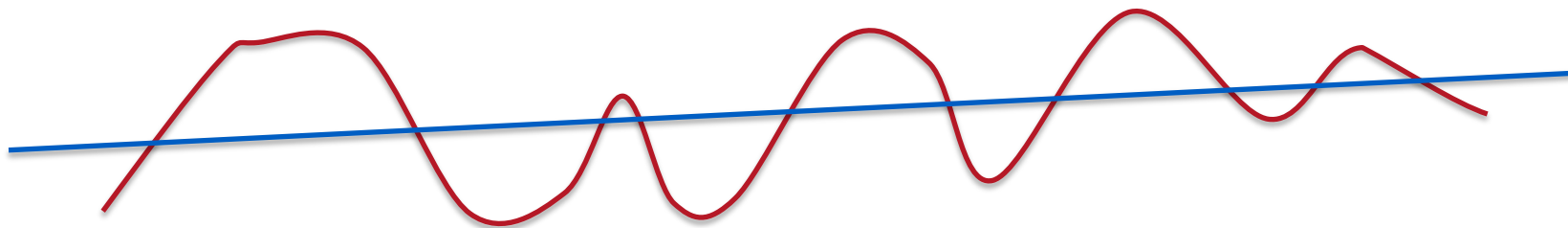


# Not Dimensionality so much as Number of Variations



(Bengio, Dellalleau & Le Roux 2007)

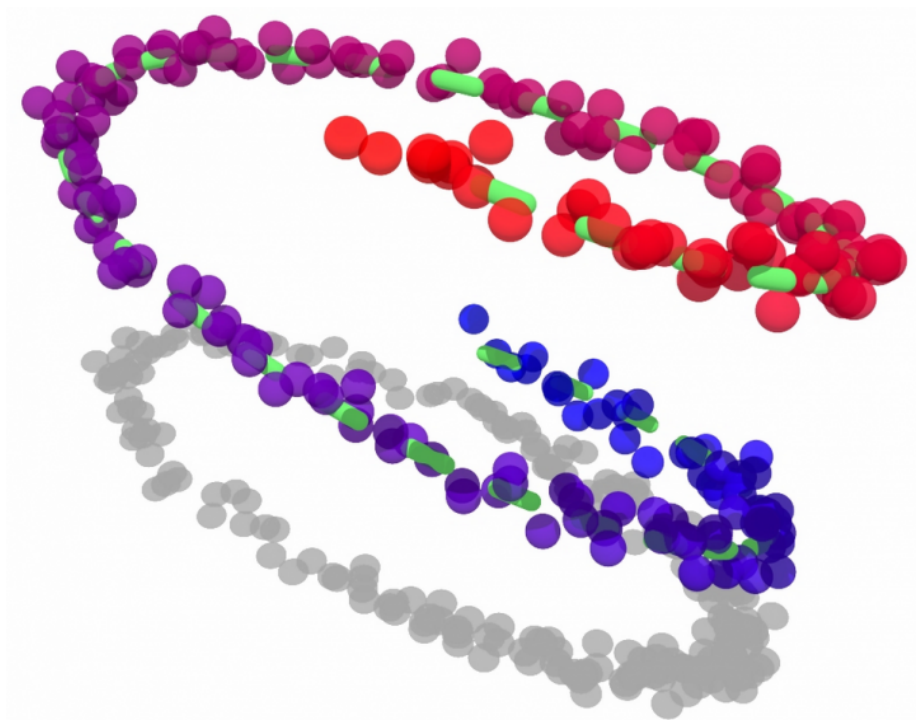
- **Theorem:** Gaussian kernel machines need at least  $k$  examples to learn a function that has  $2k$  zero-crossings along some line



- **Theorem:** For a Gaussian kernel machine to learn some maximally varying functions over  $d$  inputs requires  $O(2^d)$  examples

# Geometrical view on machine Learning

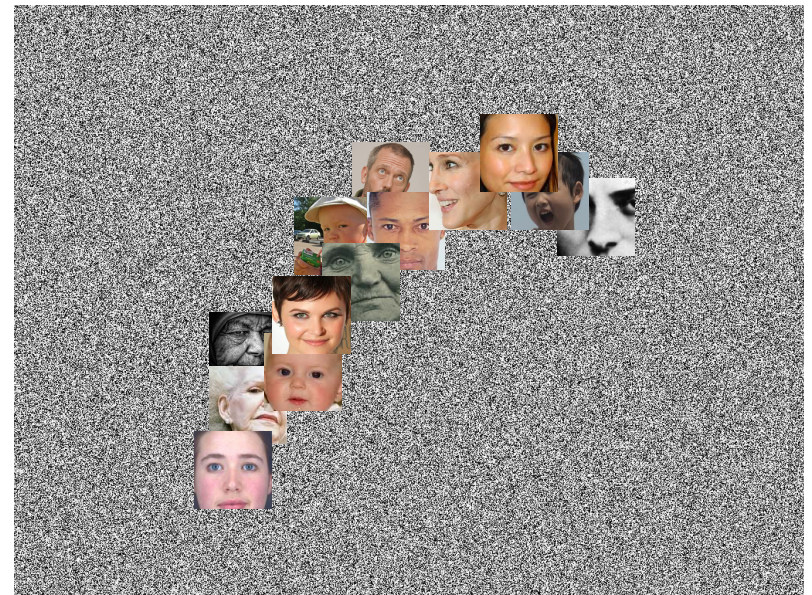
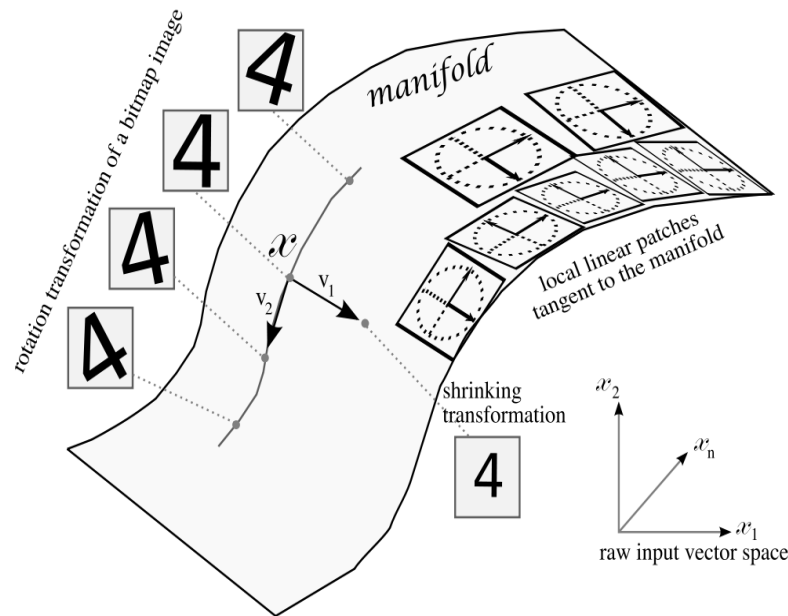
- Generalization: guessing **where** *probability* mass concentrates
- Challenge: the curse of dimensionality (exponentially many configurations of the variables to consider)



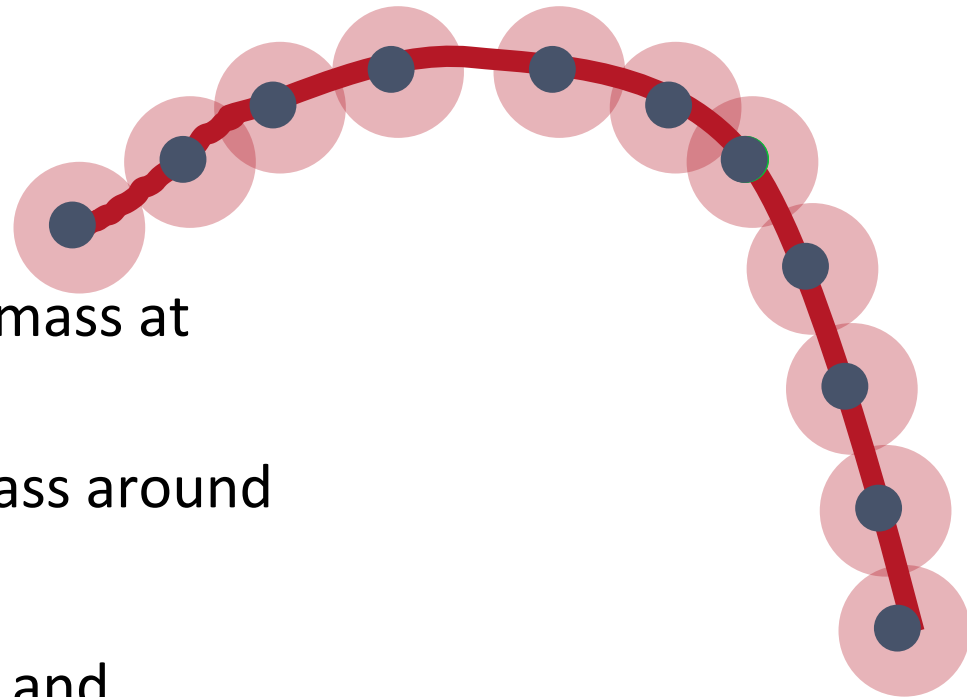


# For AI Tasks: Manifold structure

- examples **concentrate** near a lower dimensional “manifold”
- Evidence: most input configurations are unlikely



# Putting Probability Mass where Structure is Plausible



- Empirical distribution: mass at training examples
- Smoothness: spread mass around
- Insufficient
- Guess some 'structure' and generalize accordingly



Is there any hope to  
generalize non-locally?

Yes! Need good priors!

# 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**

# 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)

“Machine learning” often reduces 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**

## #2 Distributed Representations

Many neurons active simultaneously in the brain: around 1%

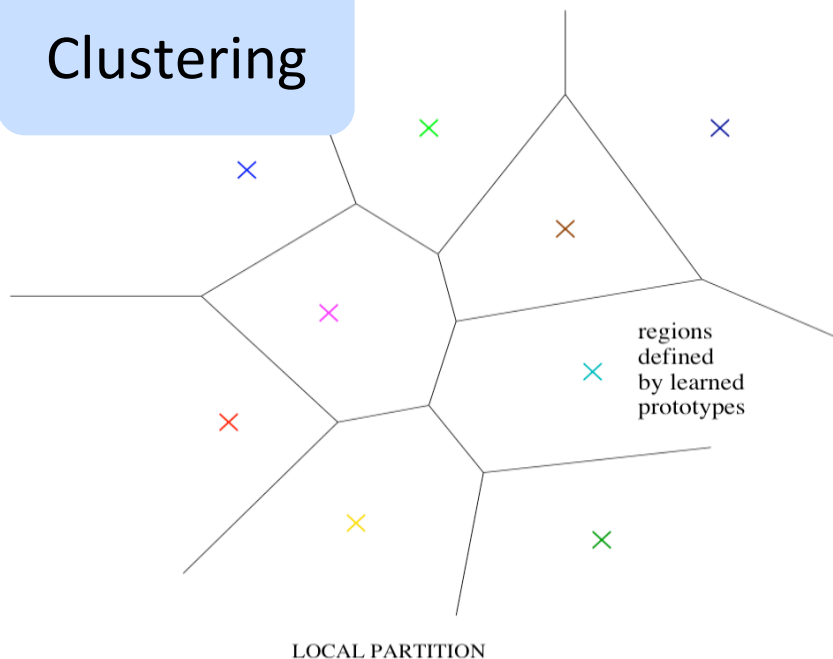
The input is represented by the activation of a set of features that are not mutually exclusive.

Can be exponentially more efficient than local representations



# #2 Non-distributed representations

## Clustering



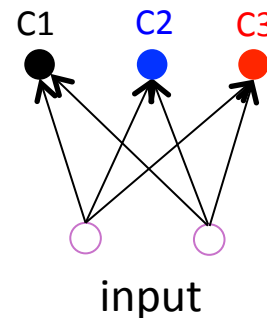
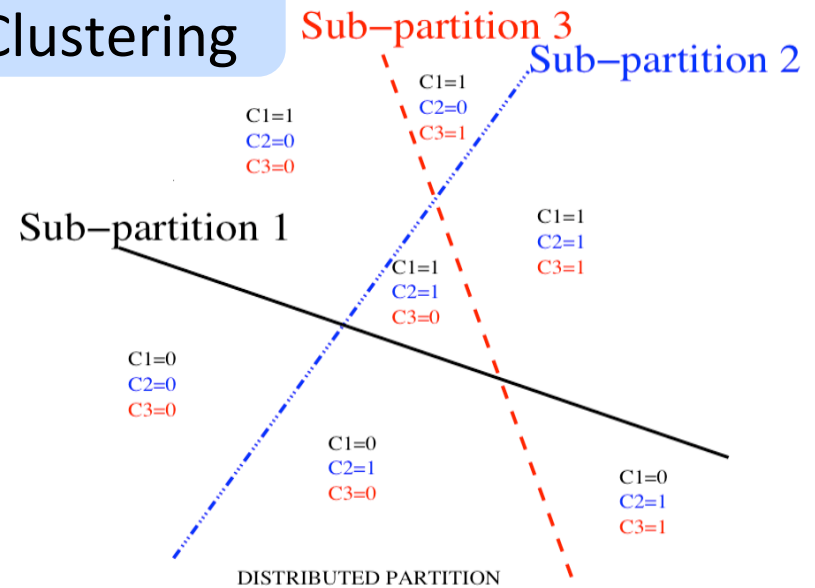
- Clustering, Nearest-Neighbors, RBF SVMs, local non-parametric density estimation & prediction, decision trees, etc.
- Parameters for each distinguishable region
- **# of distinguishable regions is linear in # of parameters**

→ No non-trivial generalization to regions without examples

# #2 The power of distributed representations

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

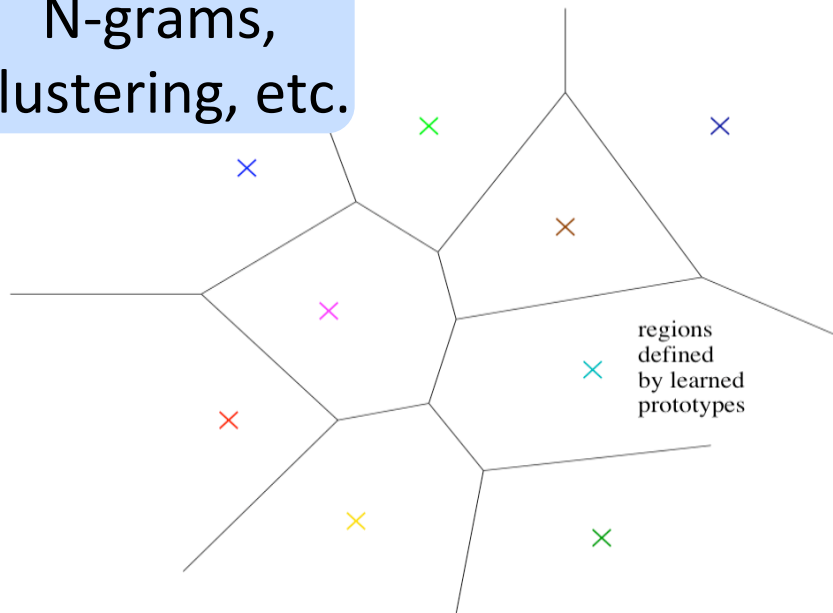
## Multi-Clustering



Non-mutually exclusive features/attributes create a combinatorially large set of distinguishable configurations

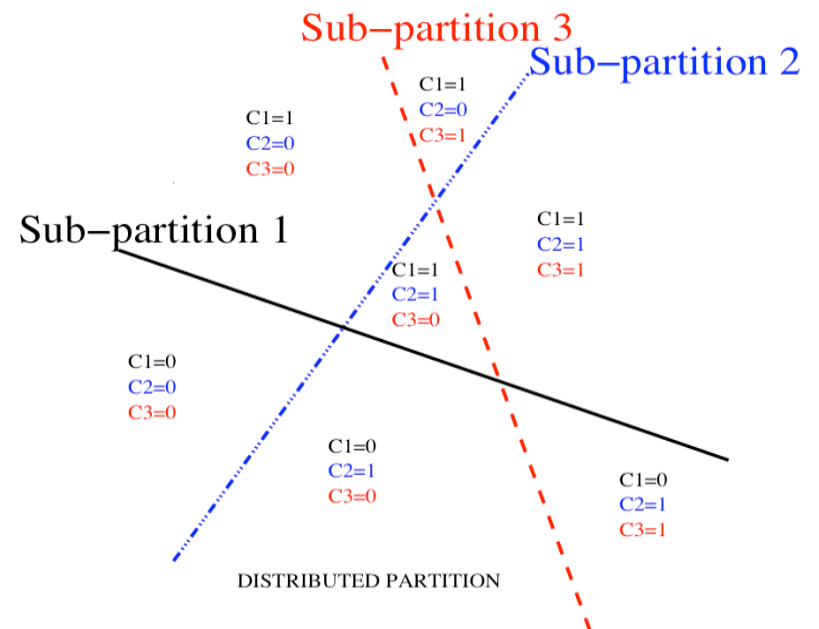
# #2 The power of distributed representations

N-grams,  
clustering, etc.



LOCAL PARTITION

Learned attributes/  
embeddings

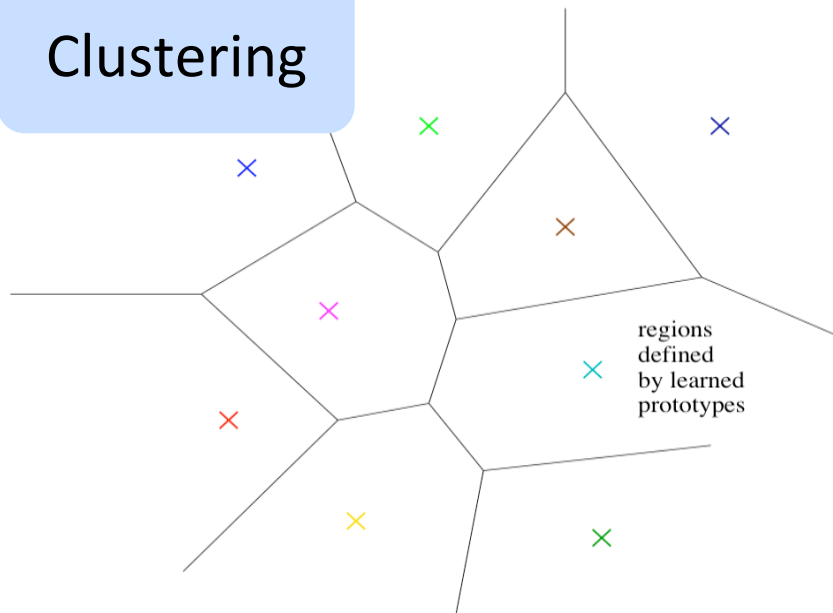


Learning a **set of features** that are not mutually exclusive can be **exponentially more statistically efficient** than having nearest-neighbor-like or clustering-like models



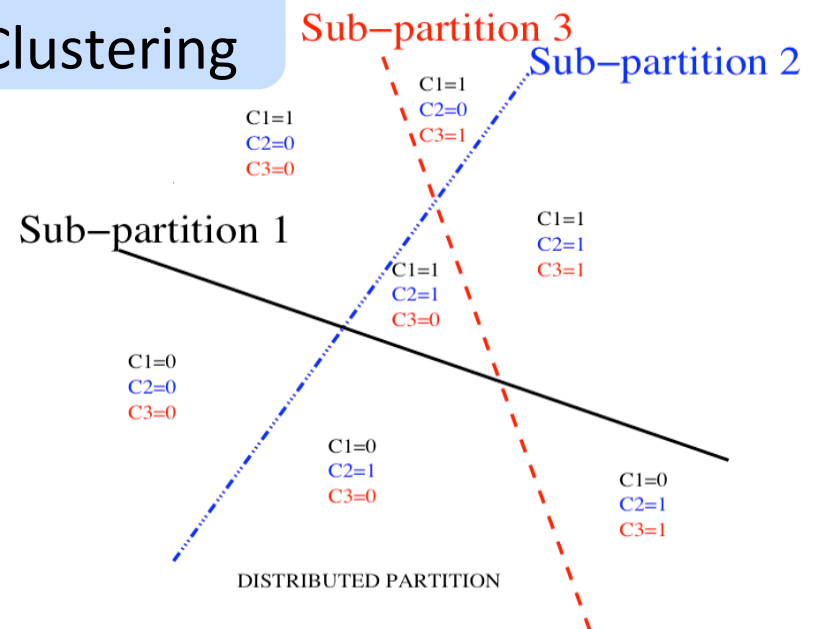
# #2 The need for distributed representations

## Clustering



LOCAL PARTITION

## Multi-Clustering



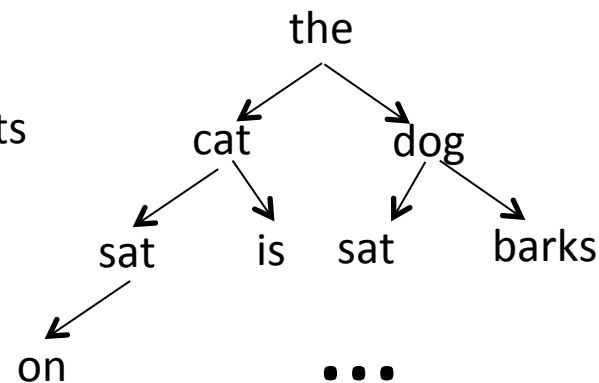
DISTRIBUTED PARTITION

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

## Why N-grams have poor generalization

- For fixed  $N$ , the function  $P(\text{next word} \mid \text{last } N-1 \text{ words})$  is learned purely from the instances of the specific  $N$ -tuples associated with each possible  $(N-1)$ -word context. No generalization to other sequences of  $N$  words.
- With back-off / smoothing models, there is some (limited) generalization arising from shorter  $n$ -grams, for which there is more data, at the price of less specific predictions.

No sharing, where lots would be possible



# #3 Unsupervised feature learning

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

But almost all **data is unlabeled**, e.g. text, images on the web

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

... in about  $10^9$  seconds

Labels cannot possibly provide enough information

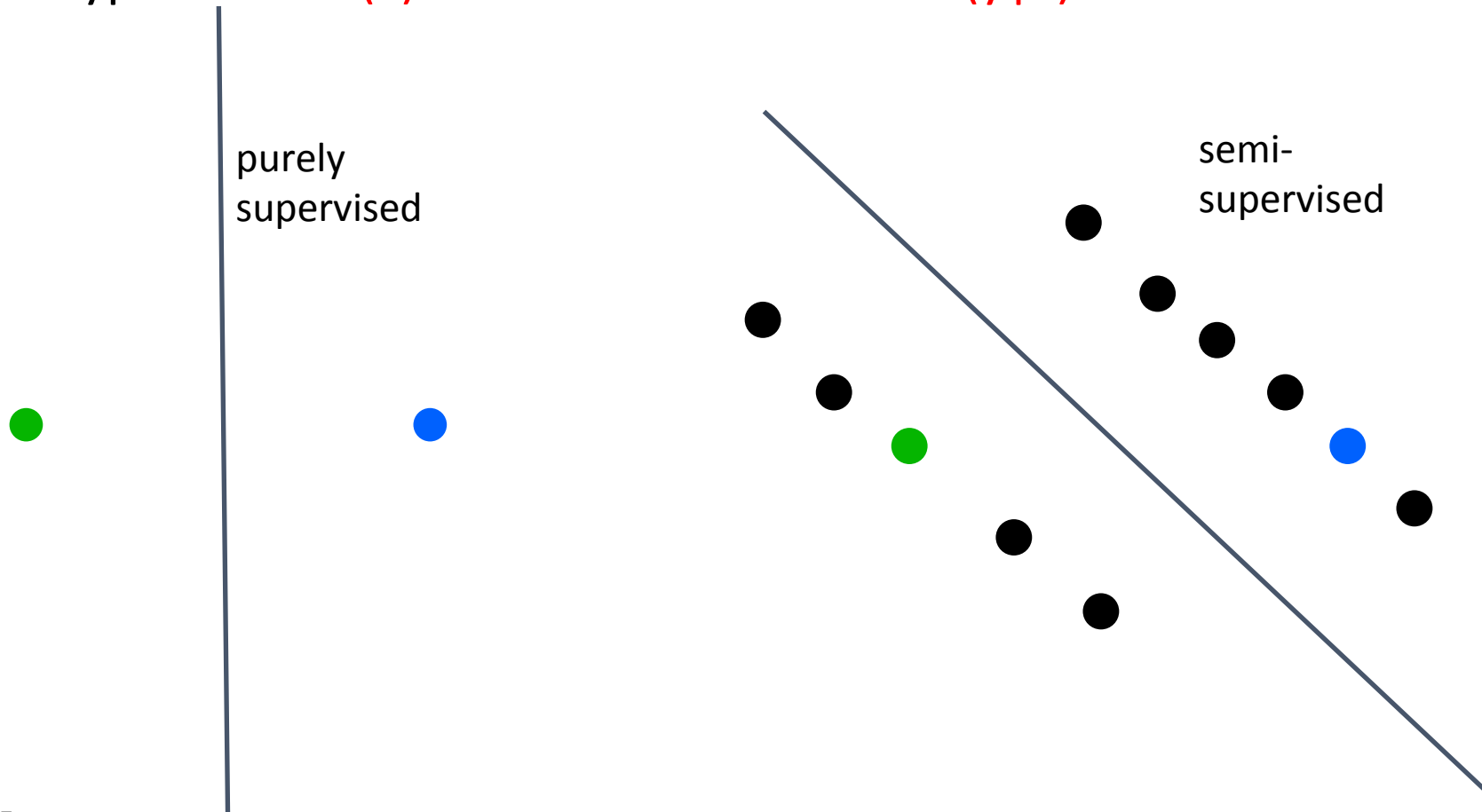
Most information acquired in an **unsupervised** fashion

# #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

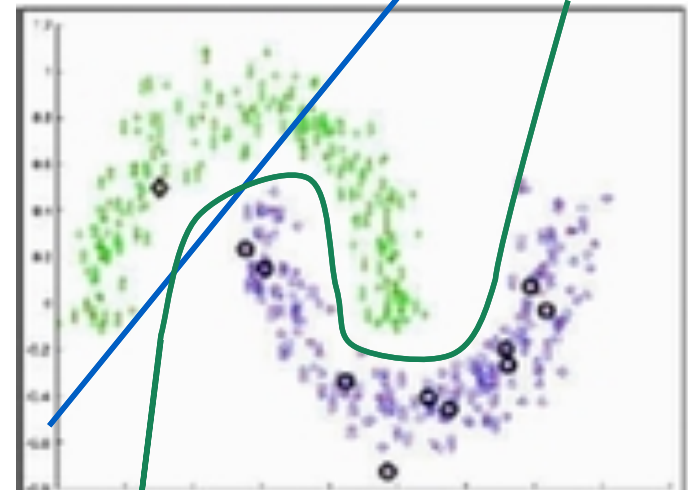
- Hypothesis:  $P(x)$  shares structure with  $P(y|x)$



# Why Semi-Supervised Learning Works

- The labeled examples (circles) help to identify the class of each cluster of unlabeled examples.
- The unlabeled examples (colored dots) help to identify the shape of each cluster.

Without unlabeled examples



With unlabeled examples

few labeled  
examples



+

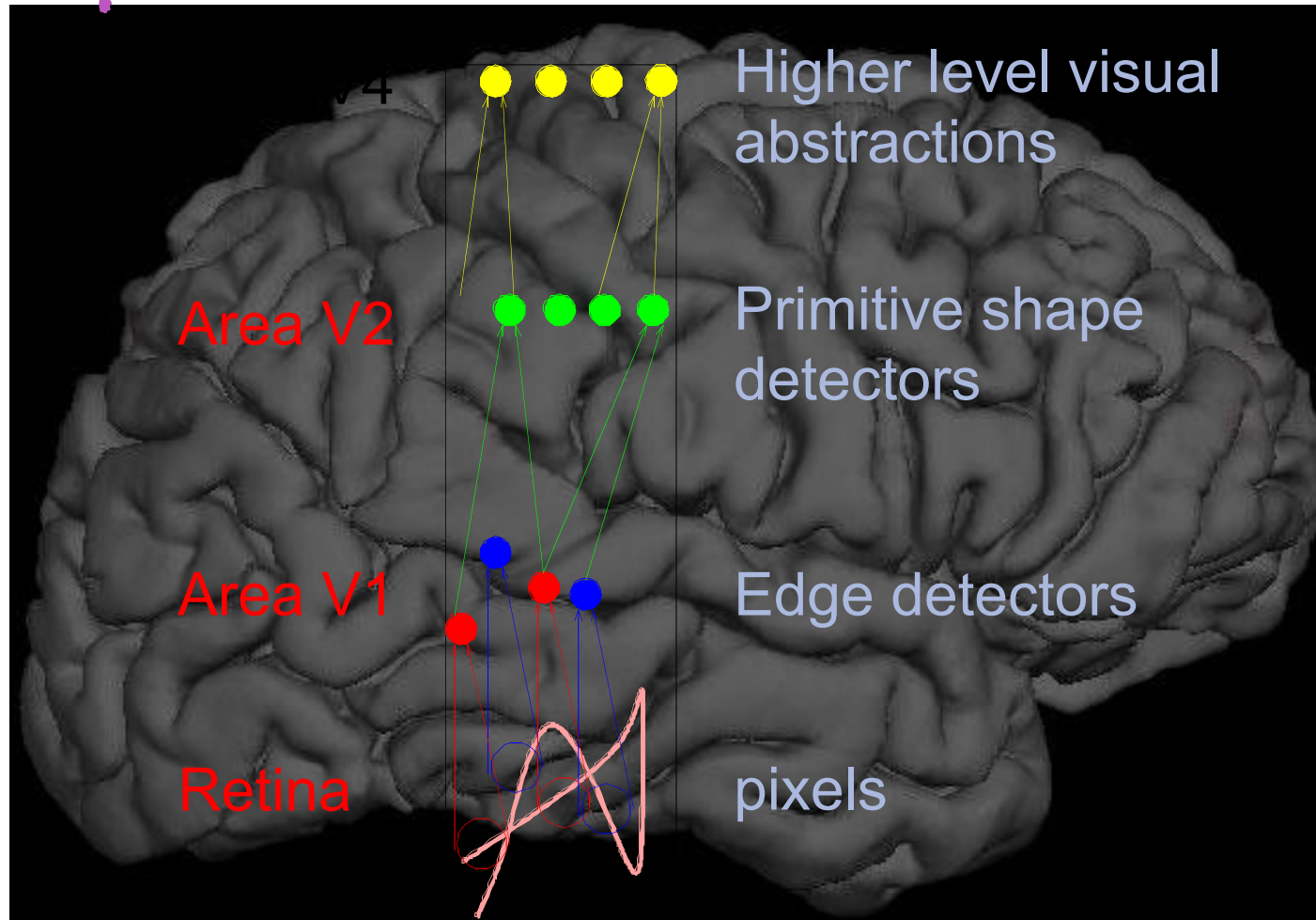


"happiness"

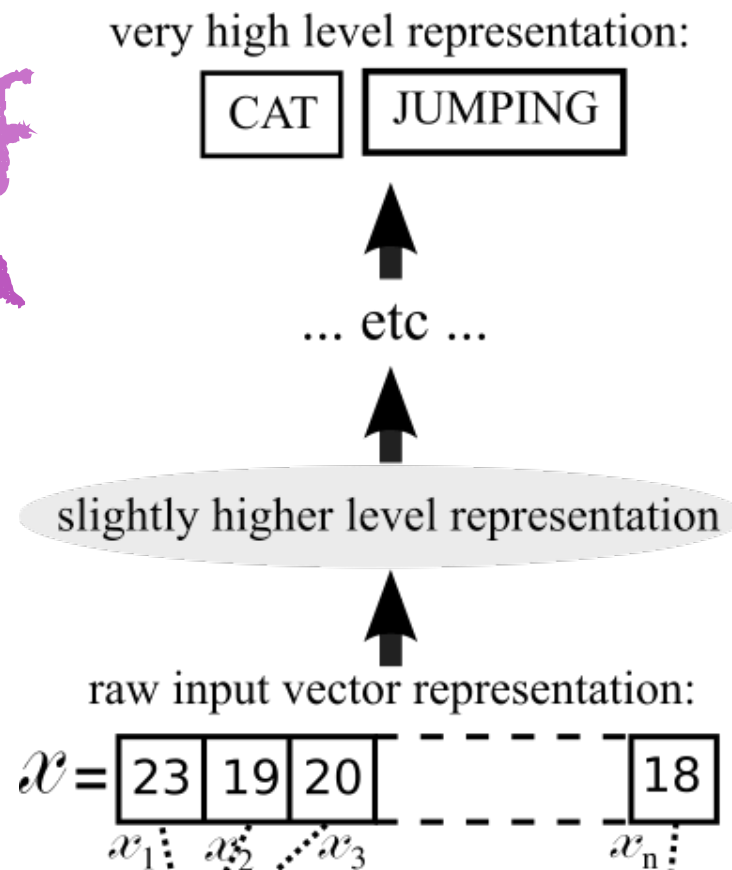
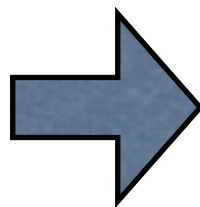
many unlabeled examples

# #4 Depth

## Deep Architecture in the Brain



# #4 Levels of Representation

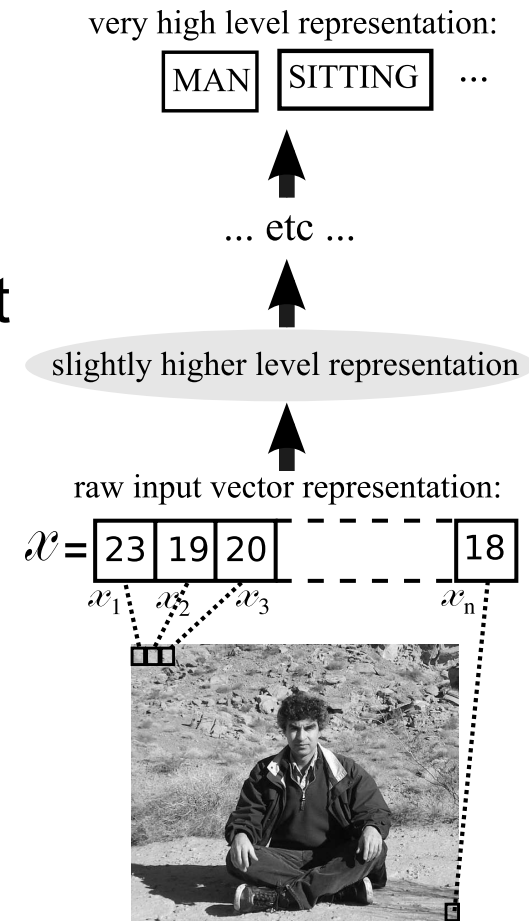




# #4 Deep Architecture in our Mind

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

**It would be good to automatically learn / discover these concepts**  
(knowledge engineering failed because of superficial introspection?)



# #4 Learning multiple levels of representation

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

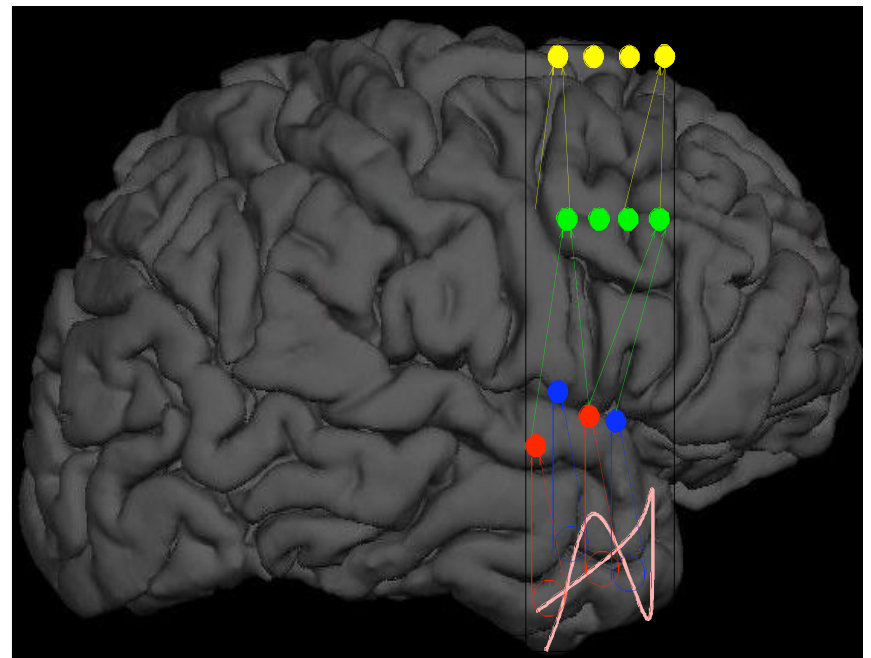
**Exponential gain for some families of functions**

Biologically inspired learning

Brain has a deep architecture

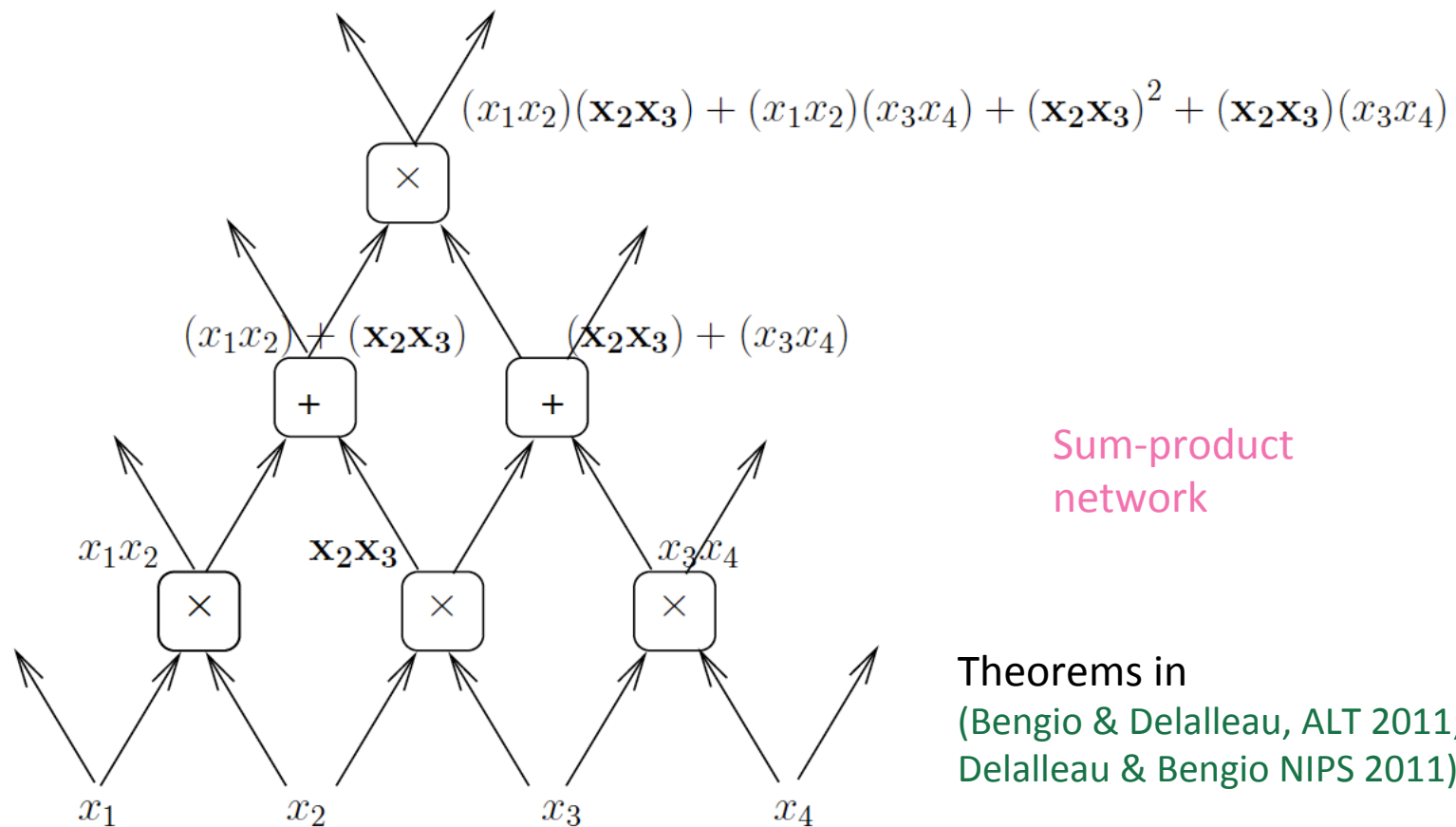
Cortex seems to have a generic learning algorithm

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



## #4 Sharing Components in a Deep Architecture

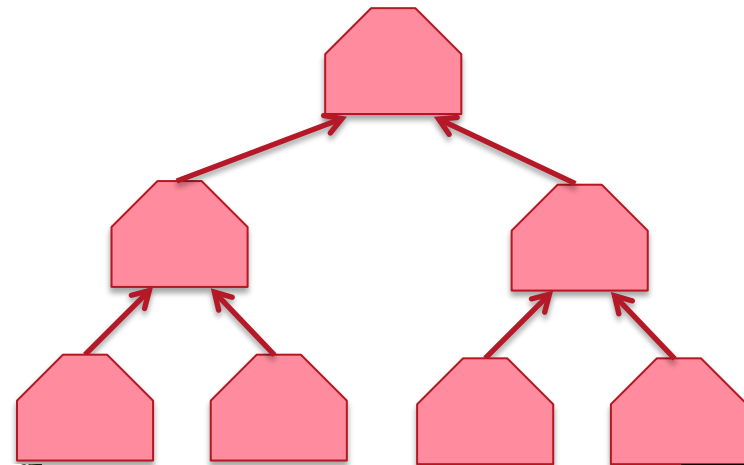
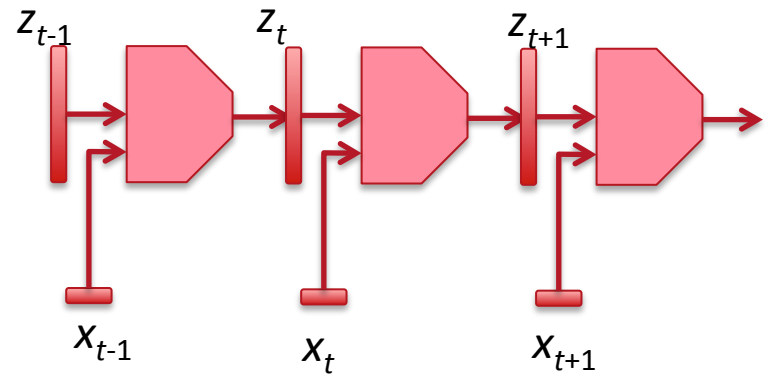
Polynomial expressed with shared components: advantage of depth may grow exponentially



Theorems in  
(Bengio & Delalleau, ALT 2011;  
Delalleau & Bengio NIPS 2011)

# #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 computation / representation

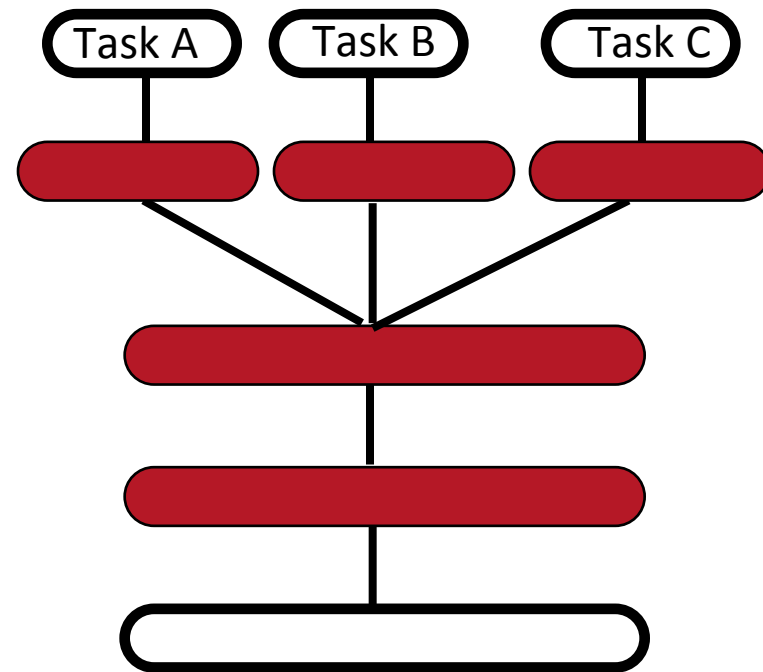


(Bottou 2011, Socher et al 2011)



# #5 Multi-Task Learning

- Generalizing better to new tasks (tens of thousands!) is crucial to approach AI
- Deep architectures learn good intermediate representations that can be shared across tasks  
(Collobert & Weston ICML 2008, Bengio et al AISTATS 2011)
- Good representations that disentangle underlying factors of variation make sense for many tasks because **each task concerns a subset of the factors**

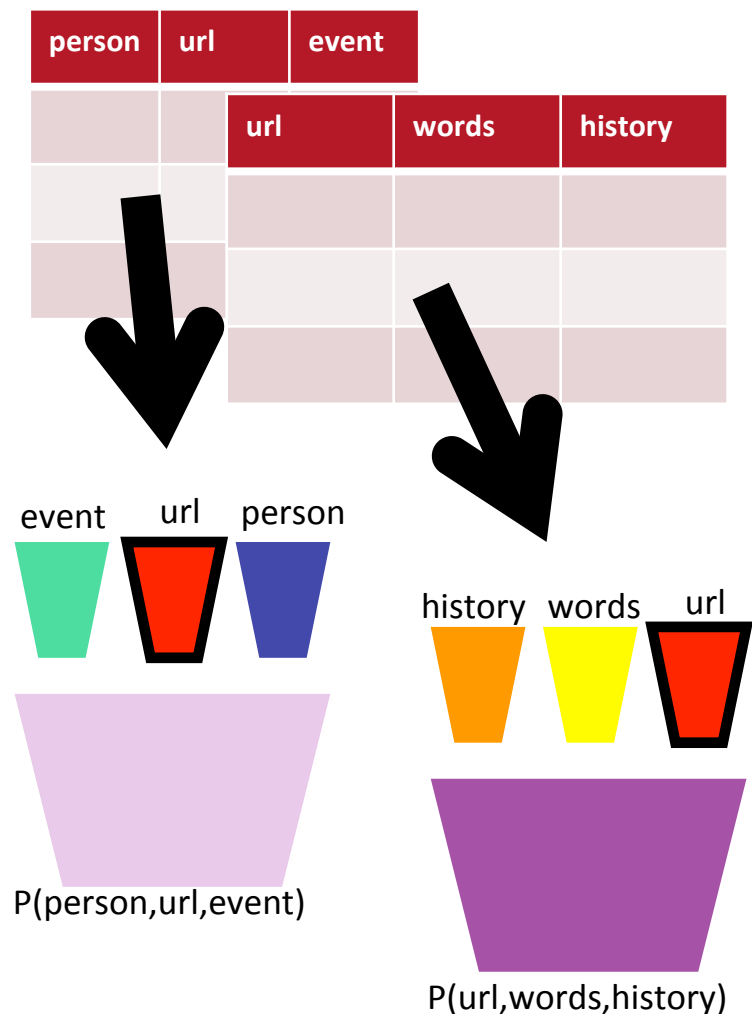


E.g. dictionary, with intermediate concepts re-used across many definitions

**Prior: 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, ML J. 2013)
- **FACTS = DATA**
- **Deduction = Generalization**



# #5 Different object types represented in same space

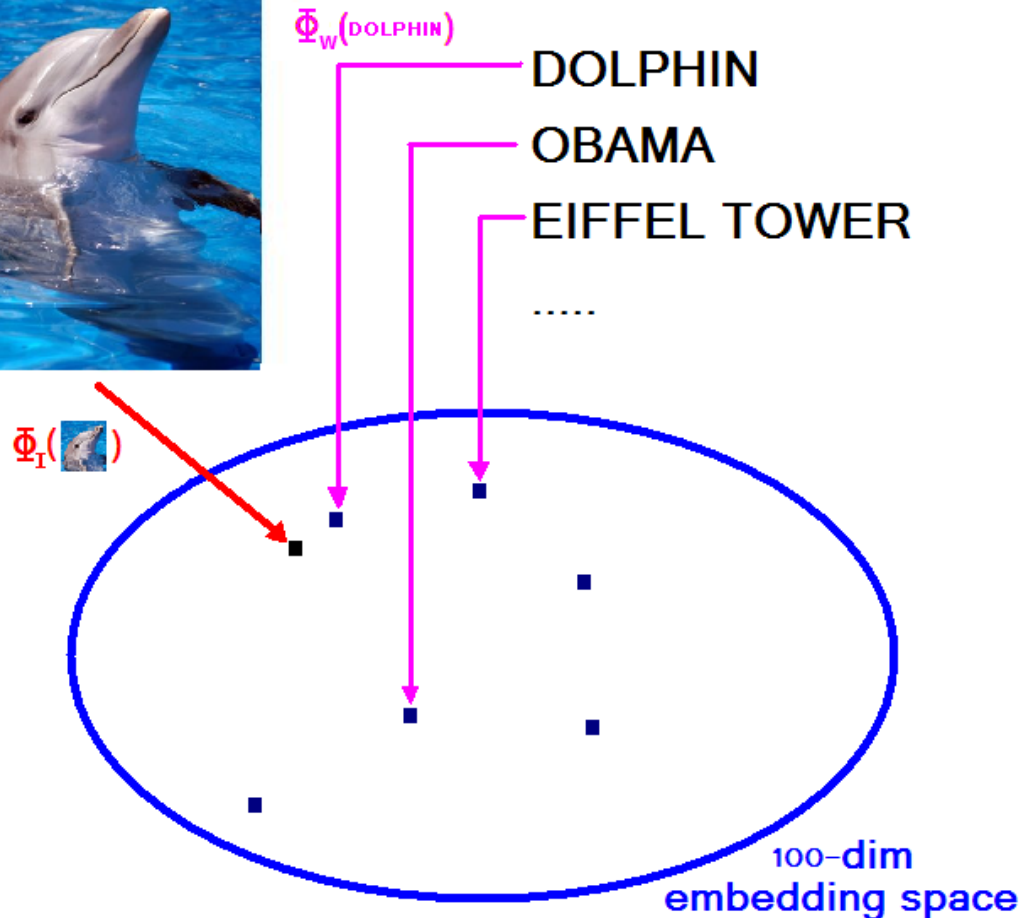


Google:

S. Bengio, J.  
Weston & N.  
Usunier



(IJCAI 2011,  
NIPS'2010,  
JMLR 2010,  
ML J. 2010)



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

# #6 Invariance and Disentangling

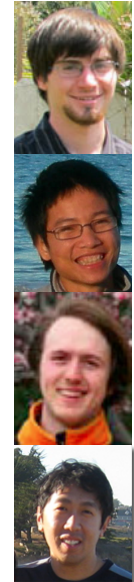
- Invariant features
- Which invariances?
- Alternative: learning to disentangle factors
- Good disentangling →  
avoid the curse of dimensionality





# #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 sparsifying penalty on learned representation (prefer 0s in the 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



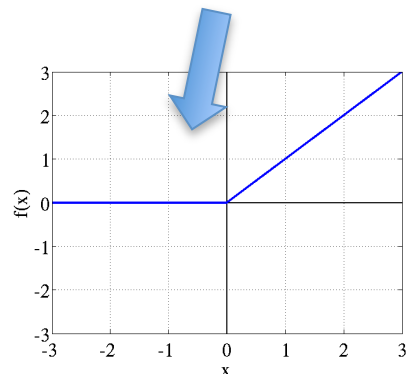
**Prior: only few concepts and attributes relevant per example**

# Deep Sparse Rectifier Neural Networks

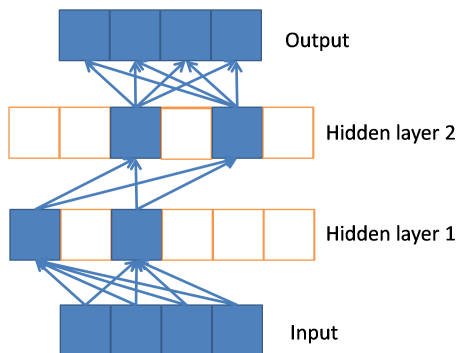
(Glorot, Bordes and Bengio AISTATS 2011), following up on (Nair & Hinton 2010) softplus RBMs

## Neuroscience motivations

Leaky integrate-and-fire model



Rectifier  
 $f(x) = \max(0, x)$



## Machine learning motivations

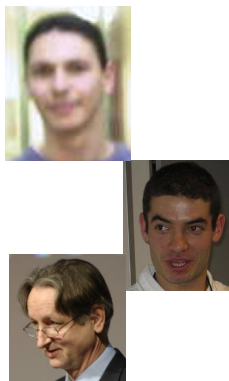
- ➡ Sparse representations
- ➡ Sparse gradients
- ➡ *Trains deep nets even w/o pretraining*



mite	container ship	motor scooter	leopard
mite	container ship	motor scooter	leopard
black widow	lifeboat	go-kart	jaguar
cockroach	amphibian	moped	cheetah
tick	fireboat	bumper car	snow leopard
starfish	drilling platform	golfcart	Egyptian cat

**Outstanding results** by Krizhevsky et al 2012  
killing the state-of-the-art on ImageNet 1000:

	1 <sup>st</sup> choice	Top-5
2 <sup>nd</sup> best		27% err
Previous SOTA	45% err	26% err
Krizhevsky et al	37% err	15% err



# Temporal Coherence and Scales

- Hints from nature about different explanatory factors:
  - Rapidly changing factors (often noise)
  - Slowly changing (generally more abstract)
  - Different factors at different time scales
- Exploit those **hints** to **disentangle** better!
- (Becker & Hinton 1993, Wiskott & Sejnowski 2002, Hurri & Hyvarinen 2003, Berkes & Wiskott 2005, Mobahi et al 2009, Bergstra & Bengio 2009)

# 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:
  - Re-use, combination and composition of learned functions/factors
  - 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 auto-encoders, sparse coding, etc.)
- New methods to successfully train deep supervised nets even without unsupervised pre-training
- Successful real-world applications, winning challenges and beating SOTAs in various areas, large-scale industrial apps

# Major Breakthrough in 2006



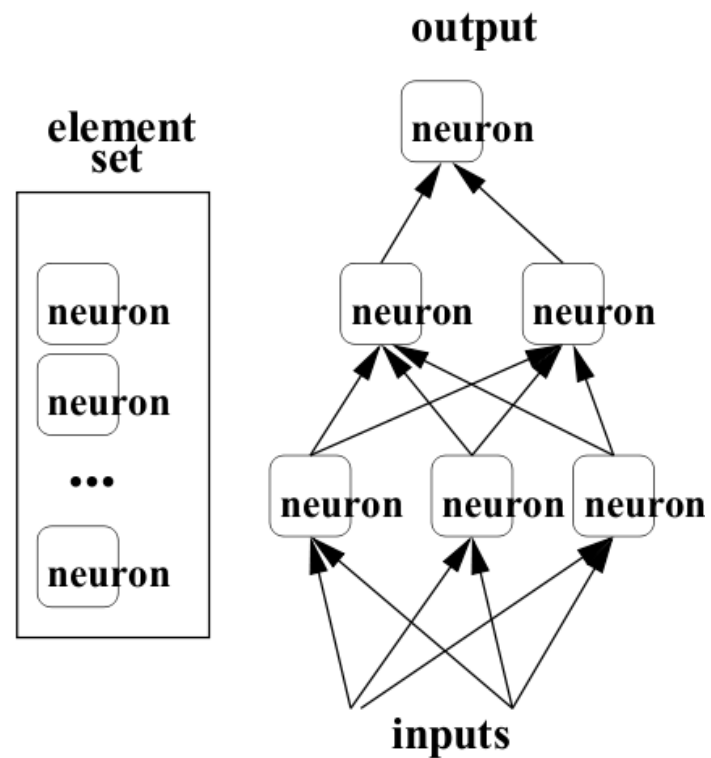
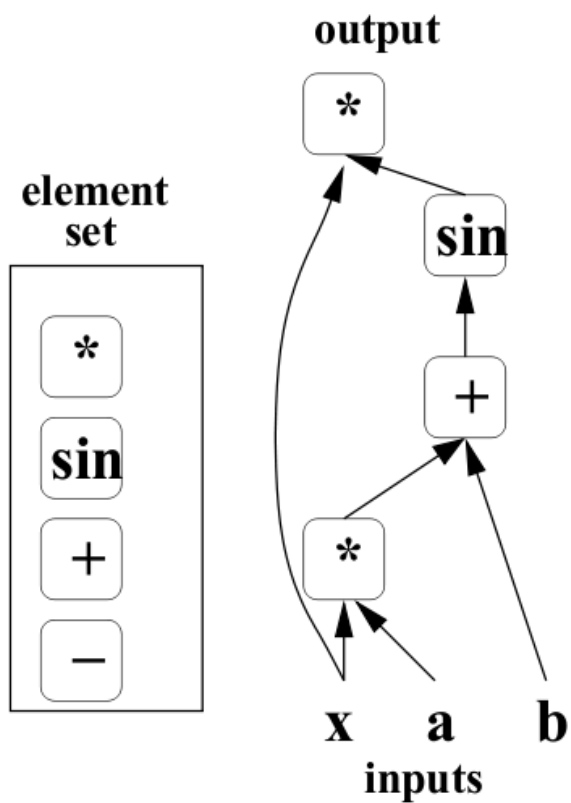
- Ability to train deep architectures by using layer-wise unsupervised learning, whereas previous purely supervised attempts had failed
- Unsupervised feature learners:
  - RBMs
  - Auto-encoder variants
  - Sparse coding variants



More about depth



# Architecture Depth



# Deep Architectures are More Expressive

Theoretical arguments:

2 layers of {  
Logic gates  
Formal neurons  
RBF units

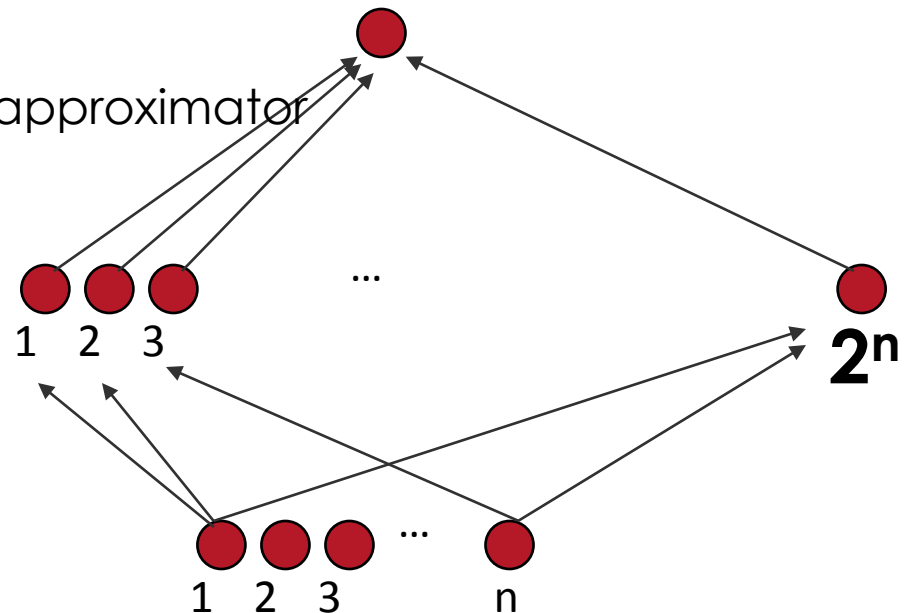
= universal approximator

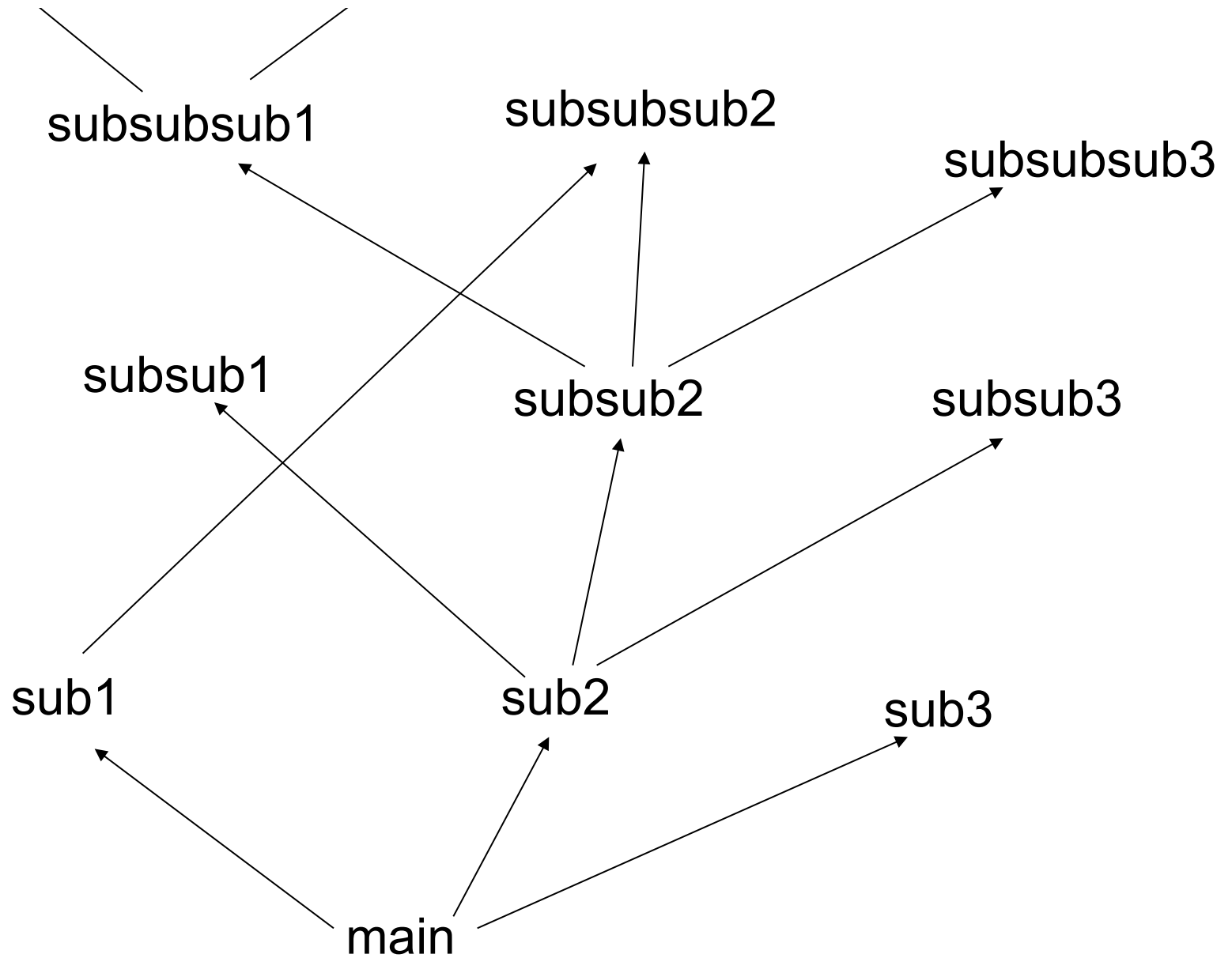
RBMs & auto-encoders = universal approximator

## Theorems on advantage of depth:

(Hastad et al 86 & 91, Bengio et al 2007, Bengio & Delalleau 2011, Braverman 2011, Pascanu et al 2014)

Some functions compactly represented with  $k$  layers may require exponential size with 2 layers

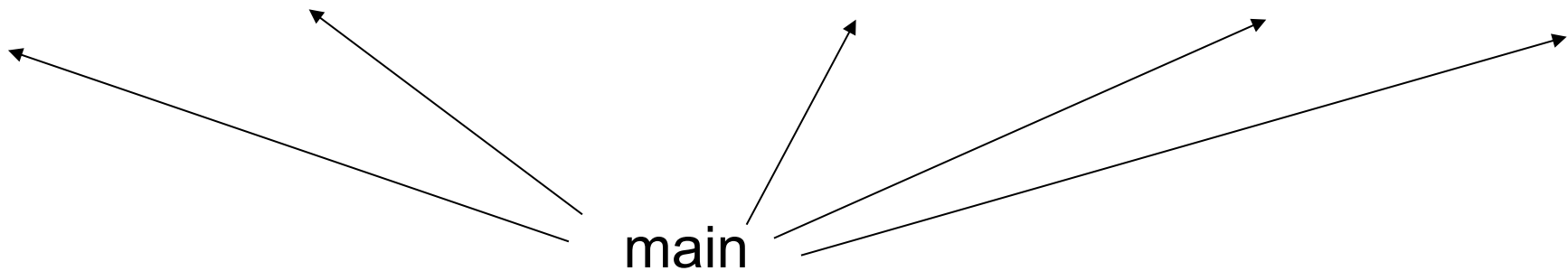




**“Deep” computer program**

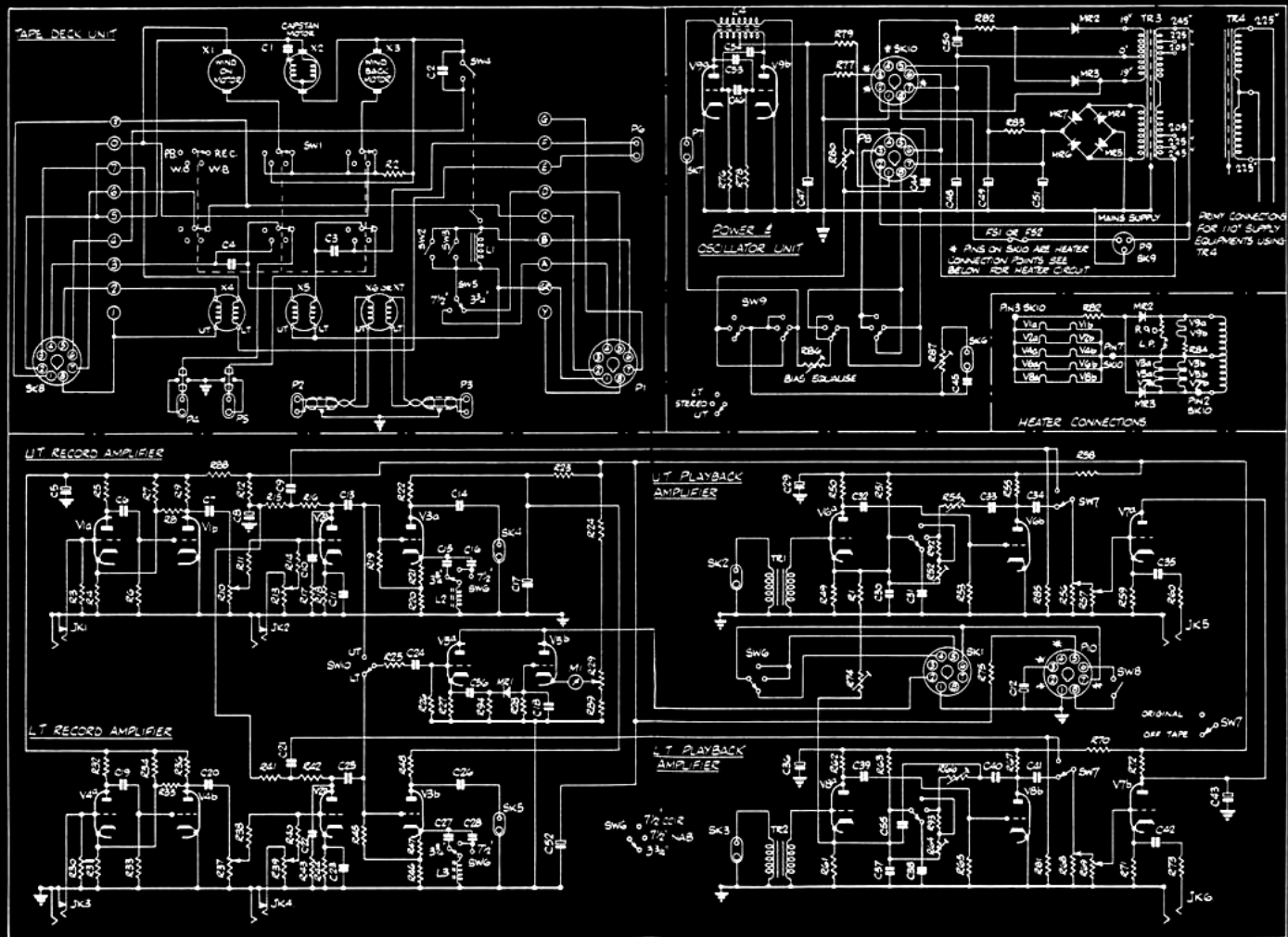
subroutine1 includes  
subsub1 code and  
subsub2 code and  
subsubsub1 code

subroutine2 includes  
subsub2 code and  
subsub3 code and  
subsubsub3 code and ...

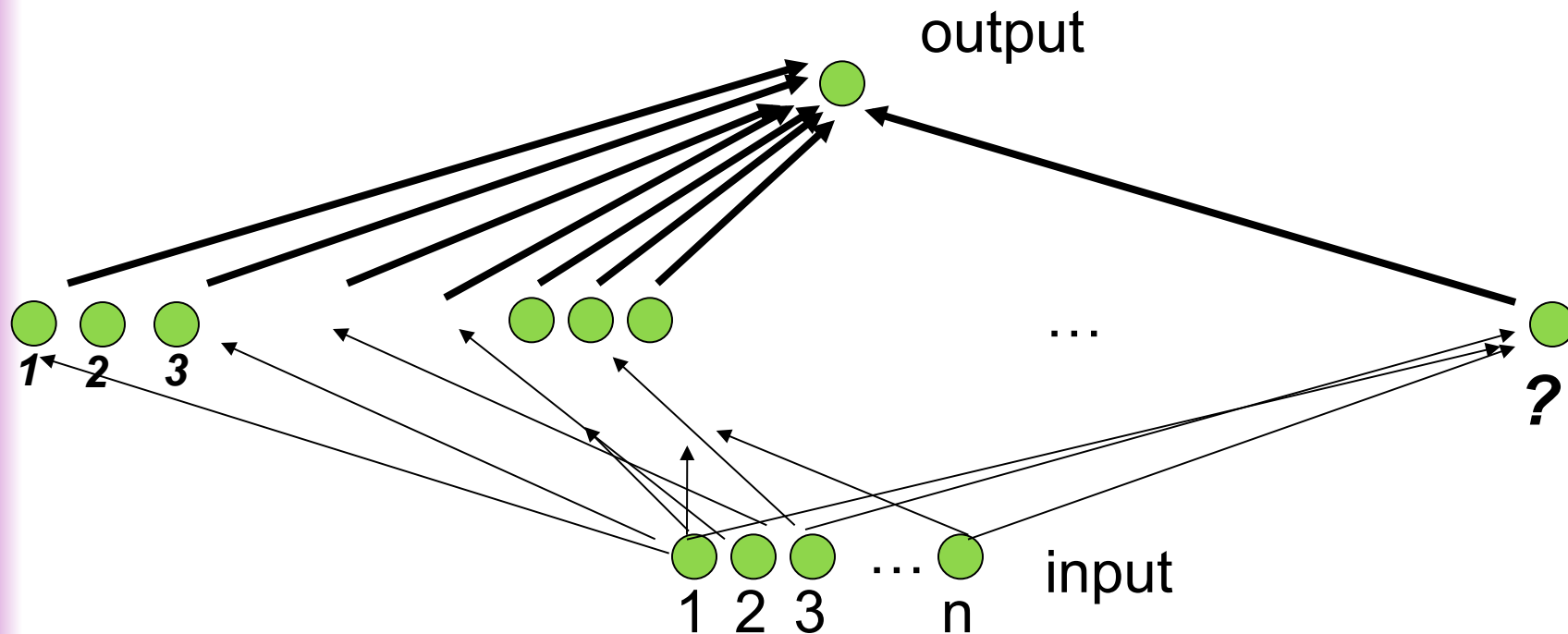


**“Shallow” computer program**

# “Deep” circuit

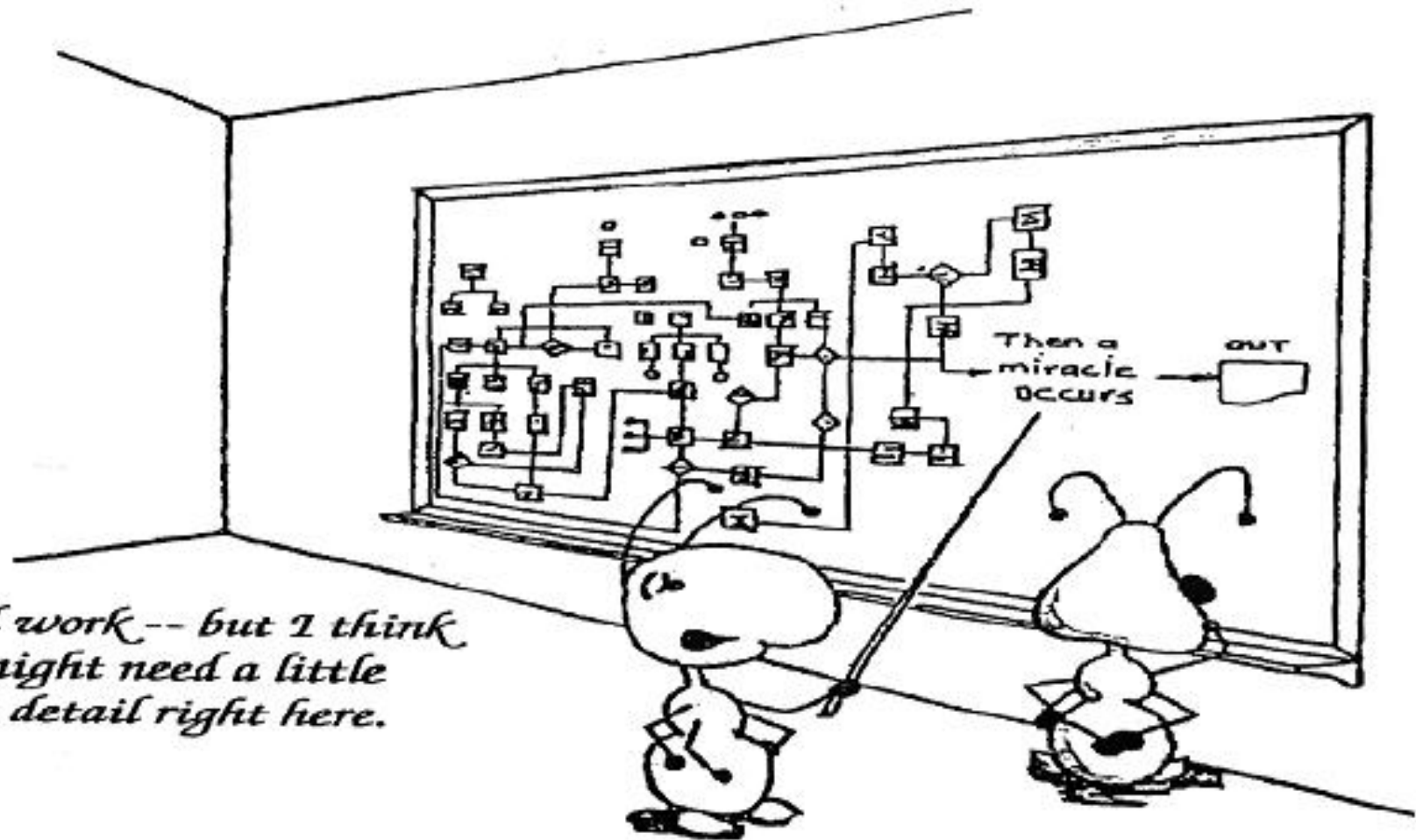


## "Shallow" circuit



Falsely reassuring theorems: one can approximate any reasonable (smooth, boolean, etc.) function with a 2-layer architecture

*Good work-- but I think  
we might need a little  
more detail right here.*



Part 2

# Representation Learning Algorithms



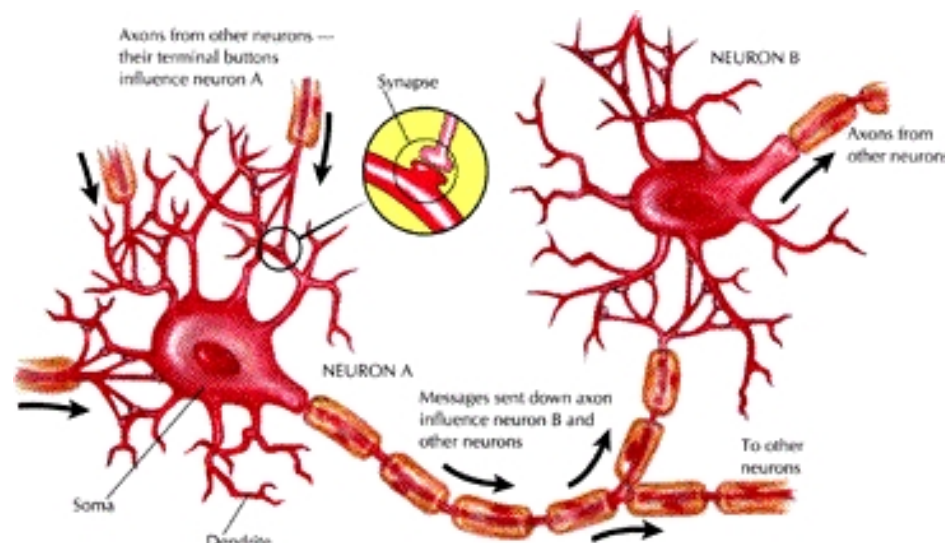
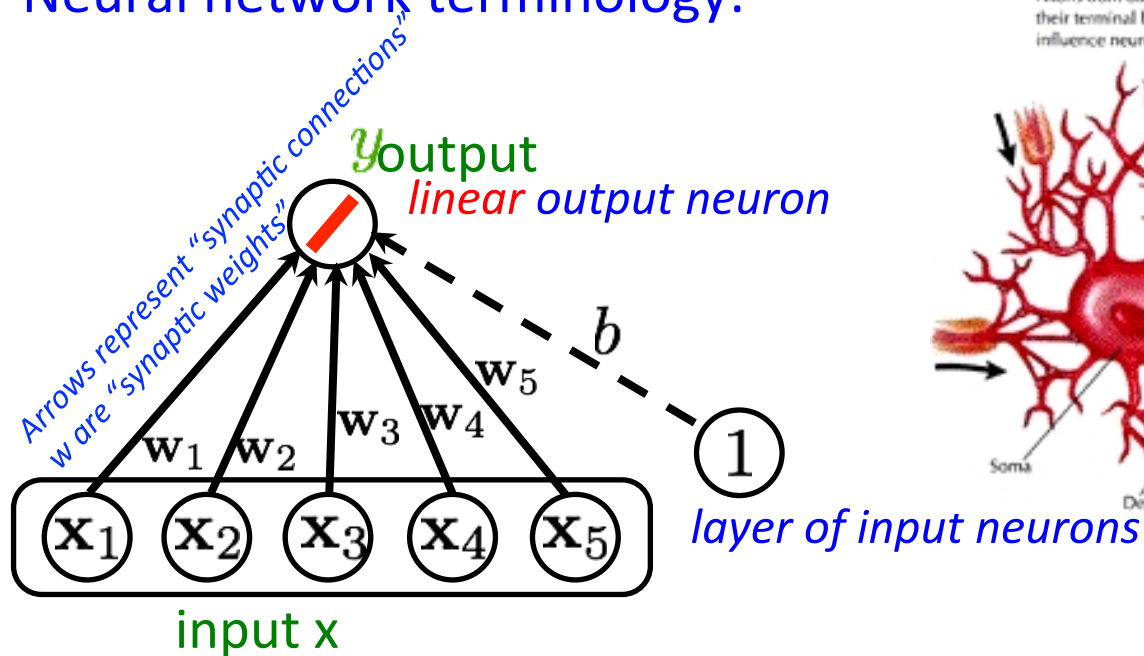
# Linear Regression

## Neural network view

Intuitive understanding of the dot product:  
each component of  $\mathbf{x}$  weighs differently on the response.

$$y = f_{\theta}(\mathbf{x}) = \mathbf{w}_1\mathbf{x}_1 + \mathbf{w}_2\mathbf{x}_2 + \dots + \mathbf{w}_d\mathbf{x}_d + b$$

Neural network terminology:



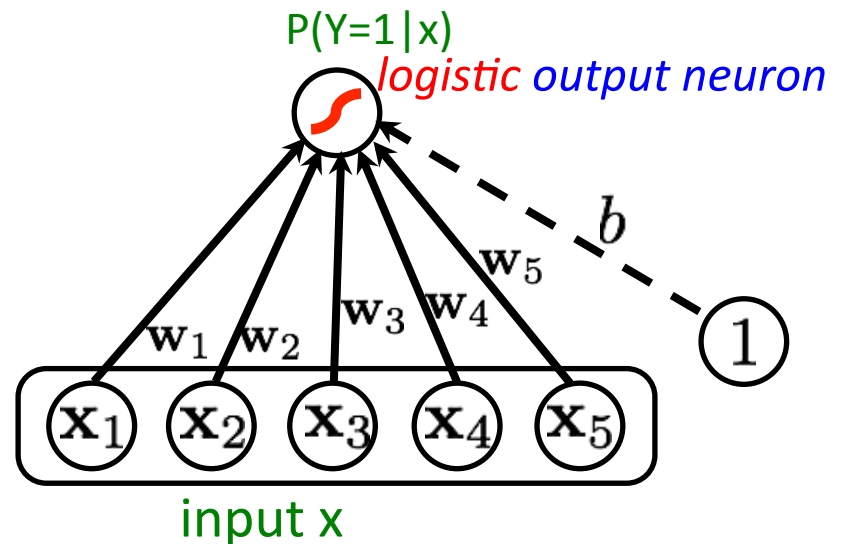
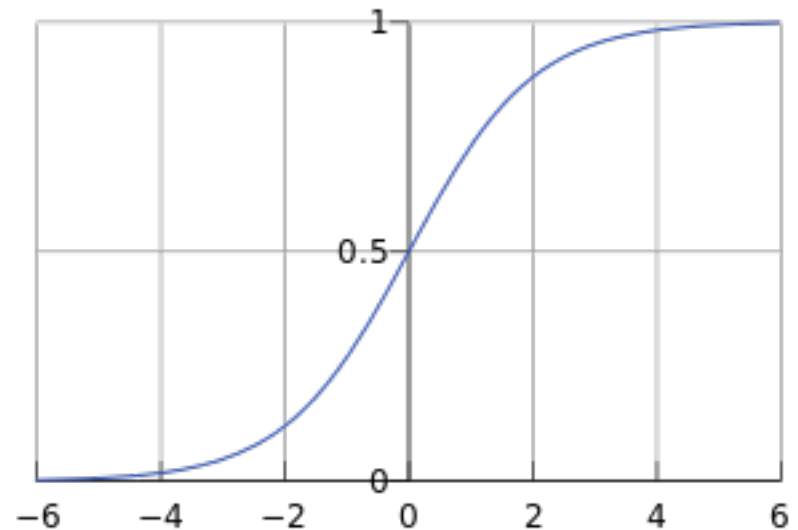
# Logistic Regression

- Predict the probability of a category  $y$ , given input  $x$ 
  - $P(Y=y \mid X=x)$
- Simple extension of linear regression (binary case):
  - $P(Y=1 \mid X=x) = \text{sigmoid}(b + w \cdot x)$
- Train by tuning  $(b, w)$  to maximize average log-likelihood

Average(  $\log P(Y=y \mid X=x)$  )

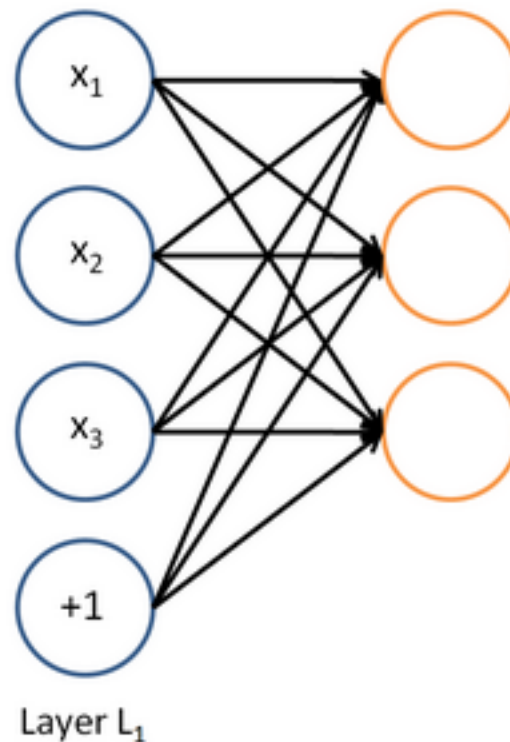
over training pairs  $(x, y)$ , by gradient-based optimization

- This is a very shallow neural network (no hidden layer)



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

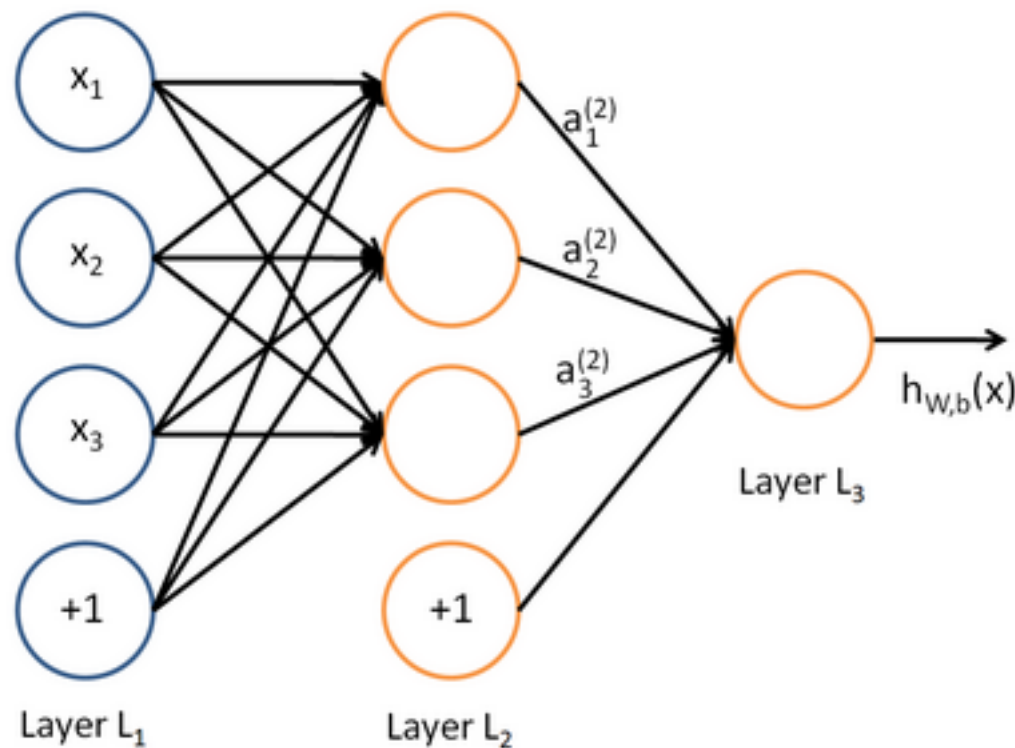
If we feed a vector of inputs through a bunch of logistic regression functions, then we get a vector of outputs



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

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

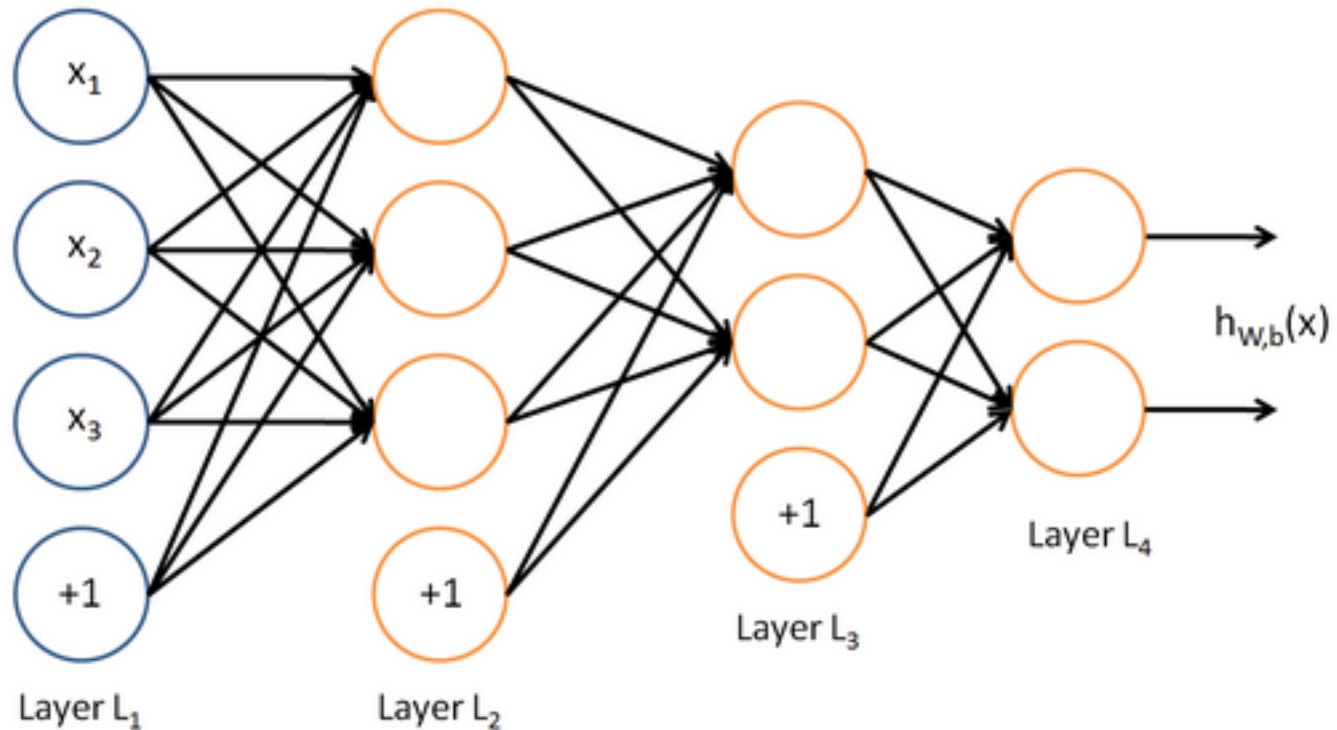
... which we can feed into another logistic regression function



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

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

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



# Back-Prop

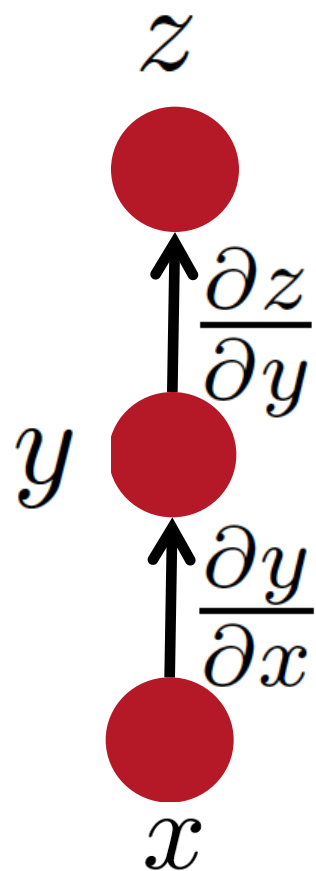
- Compute gradient of example-wise loss wrt parameters

- Simply applying the derivative chain rule wisely

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

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

## Simple Chain Rule



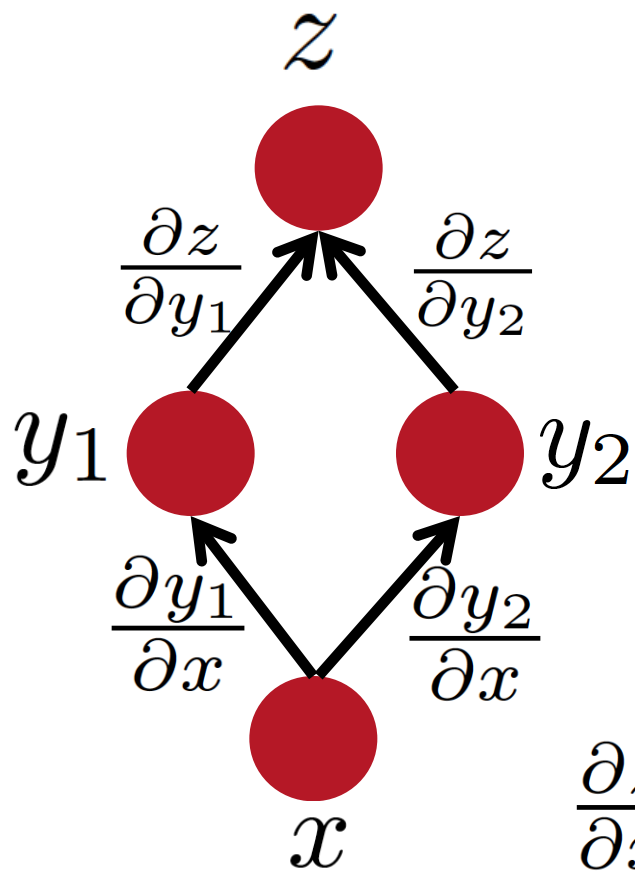
$$\Delta z = \frac{\partial z}{\partial y} \Delta y$$

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

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

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

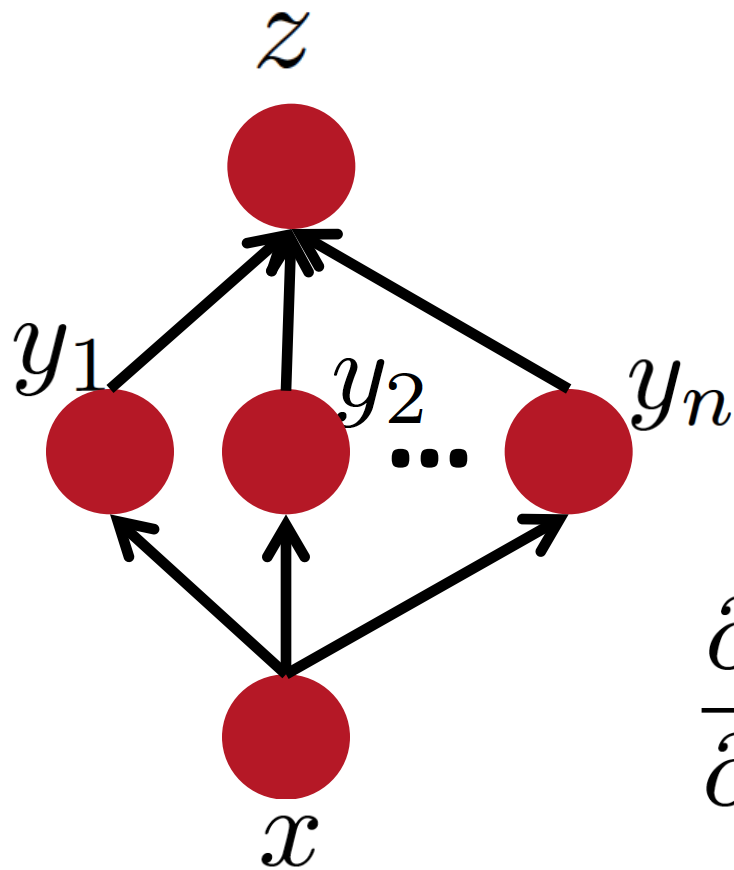
## Multiple Paths Chain Rule



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

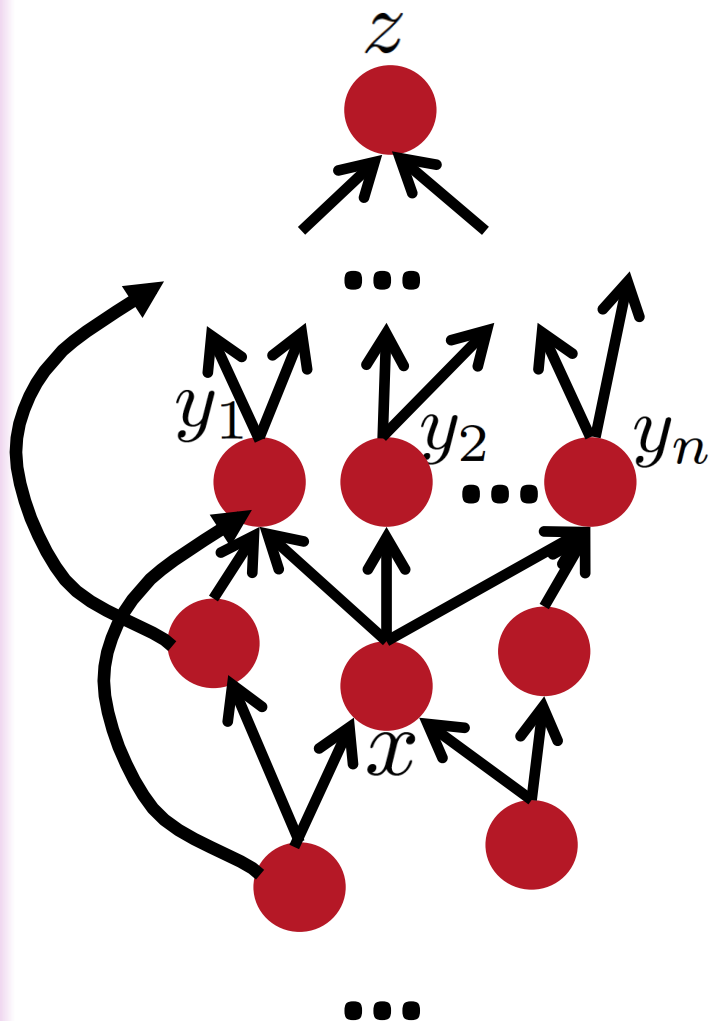


## Multiple Paths Chain Rule - General



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

# Chain Rule in Flow Graph

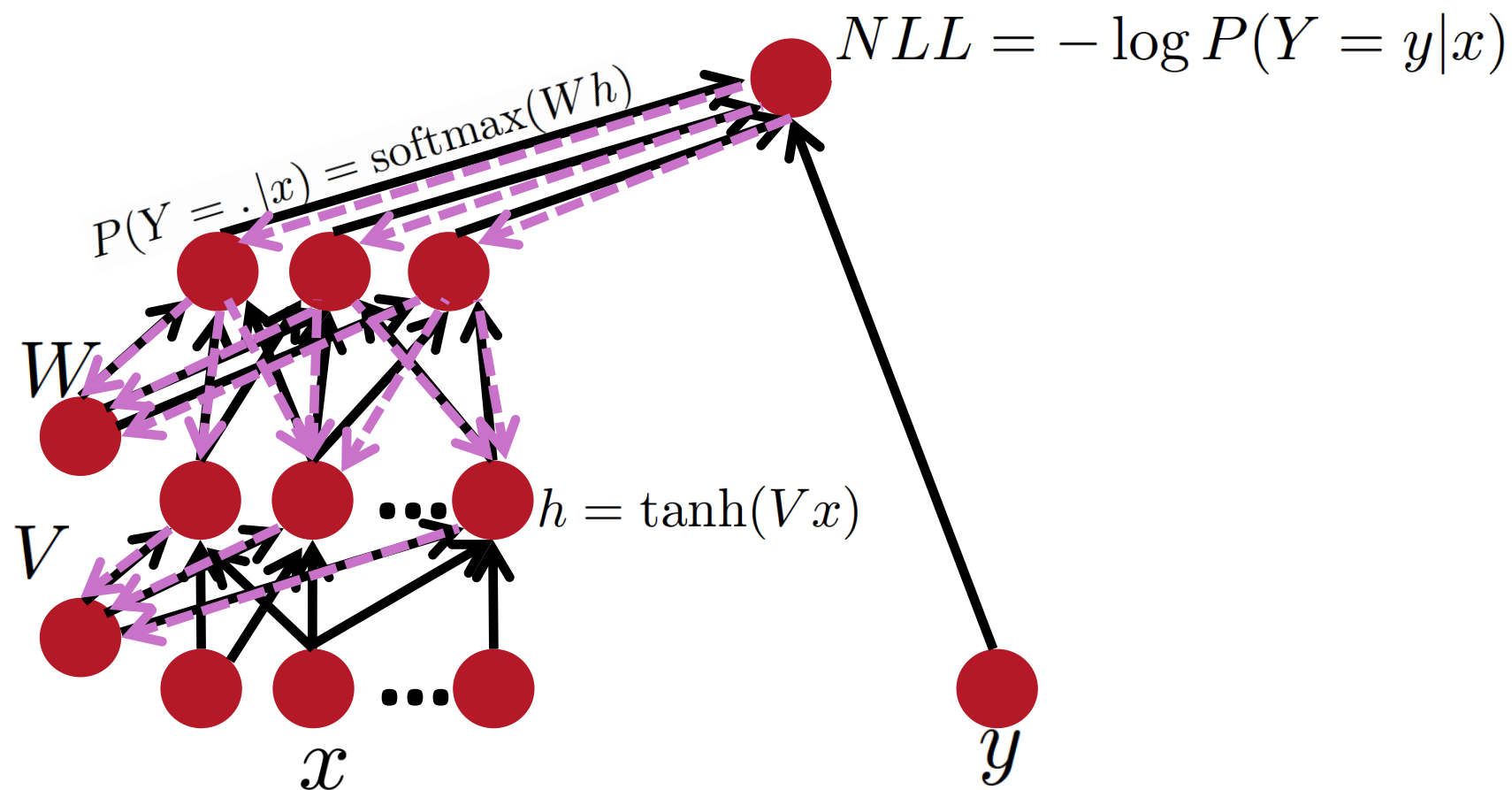


Flow graph: any directed acyclic graph  
 node = computation result  
 arc = computation dependency

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

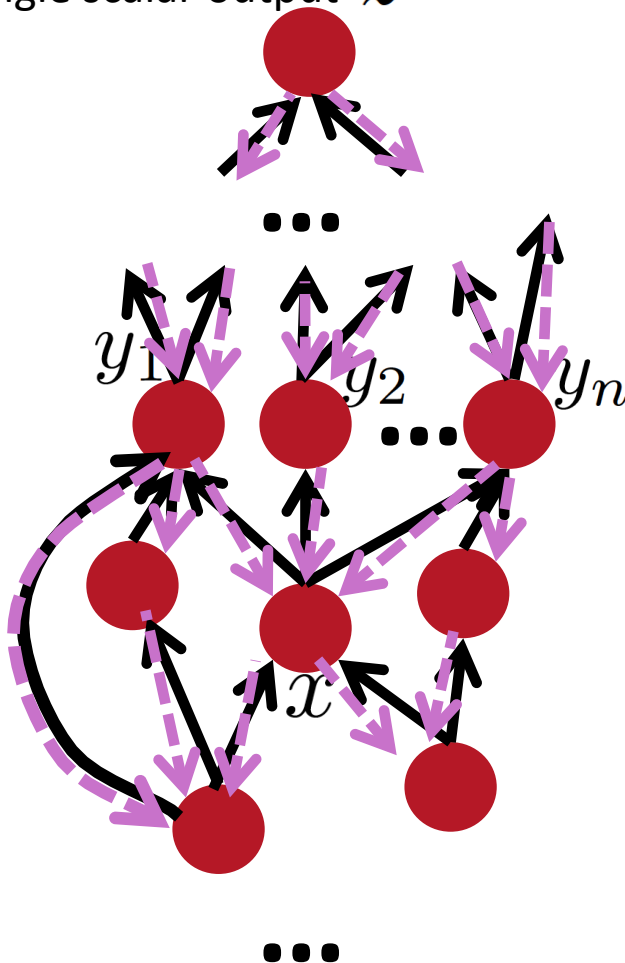
$$\frac{\partial z}{\partial x} = \sum_{i=1}^n \frac{\partial z}{\partial y_i} \frac{\partial y_i}{\partial x}$$

# Back-Prop in Multi-Layer Net



# Back-Prop in General Flow Graph

Single scalar output  $z$



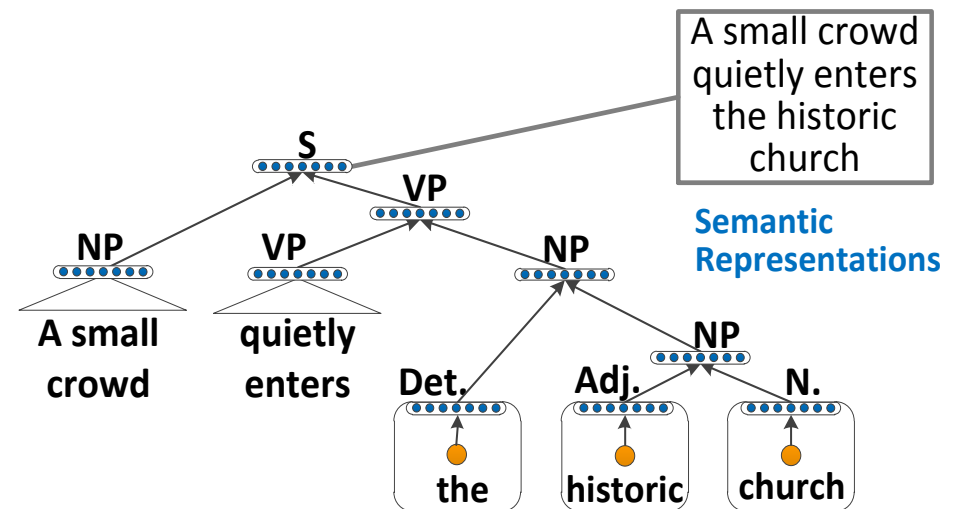
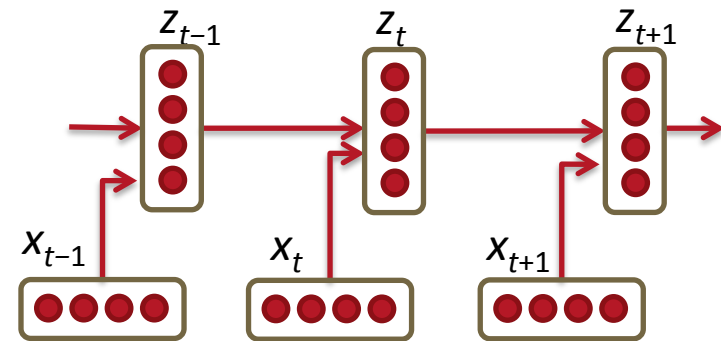
1. Fprop: visit nodes in topo-sort order
  - Compute value of node given predecessors
2. Bprop:
  - initialize output gradient = 1
  - visit nodes in reverse order:
    - Compute gradient wrt each node using gradient wrt successors

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

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

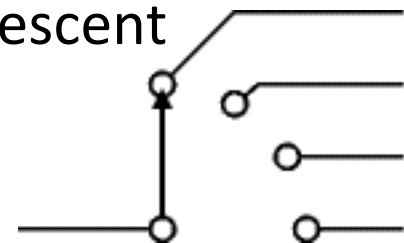
# Back-Prop in Recurrent & Recursive Nets

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

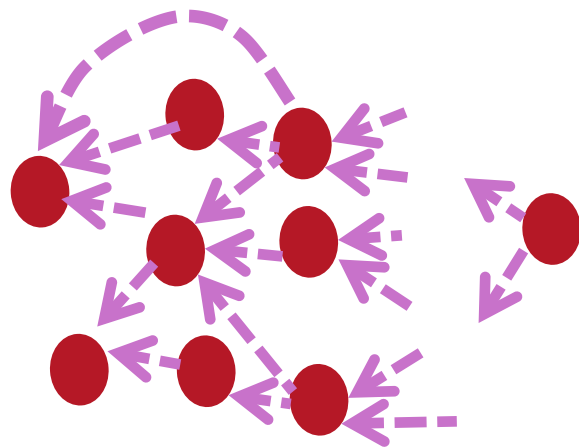
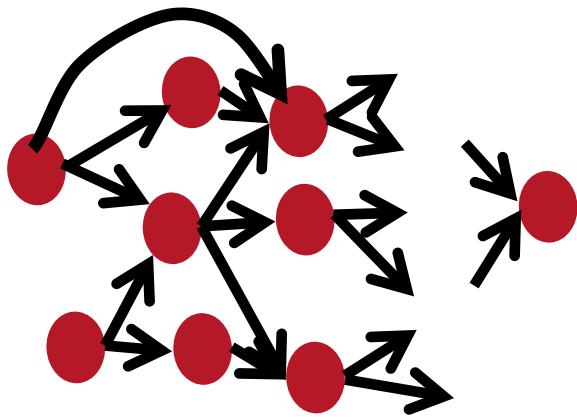


# Backpropagation Through Structure

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



# Automatic Differentiation

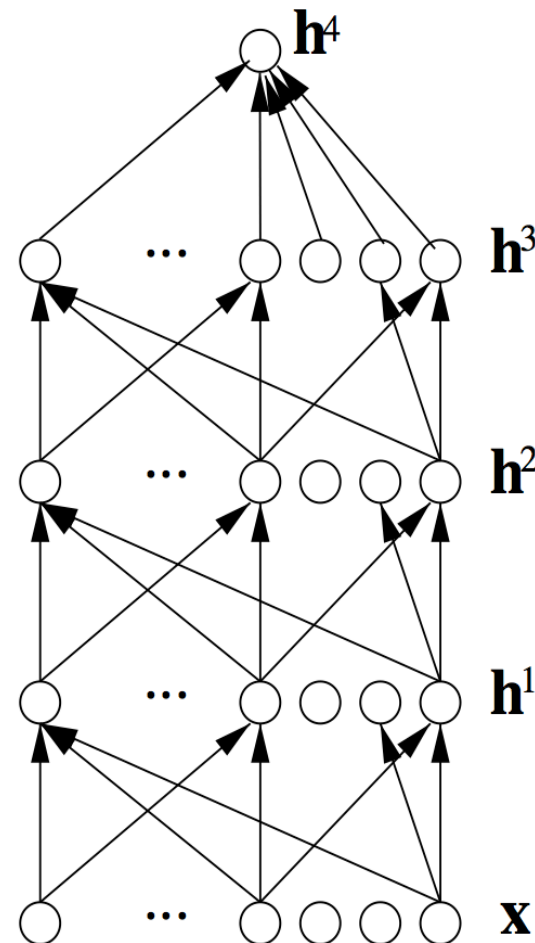


- The gradient computation can be automatically inferred from the symbolic expression of the fprop.
- Each node type needs to know how to compute its output and how to compute the gradient wrt its inputs given the gradient wrt its output.
- Easy and fast prototyping

theano

# Deep Supervised Neural Nets

- We can now train them even without unsupervised pre-training, thanks to better initialization and non-linearities (rectifiers, maxout) and they can generalize well with large labeled sets and dropout.
- Unsupervised pre-training still useful for rare classes, transfer, smaller labeled sets, or as an extra regularizer.



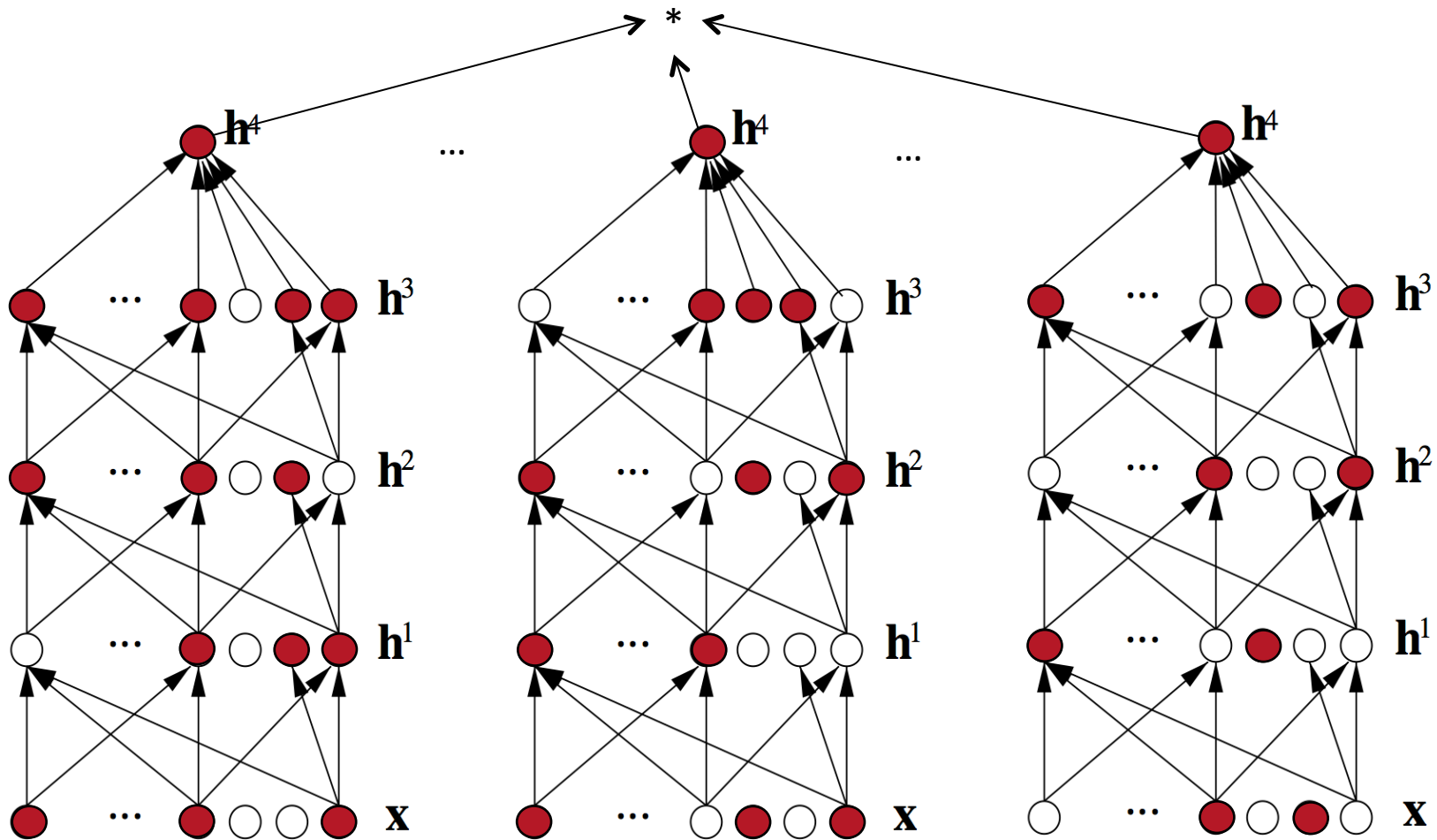


# Stochastic Neurons as Regularizer:

Improving neural networks by preventing co-adaptation of feature detectors (Hinton et al 2012, arXiv)

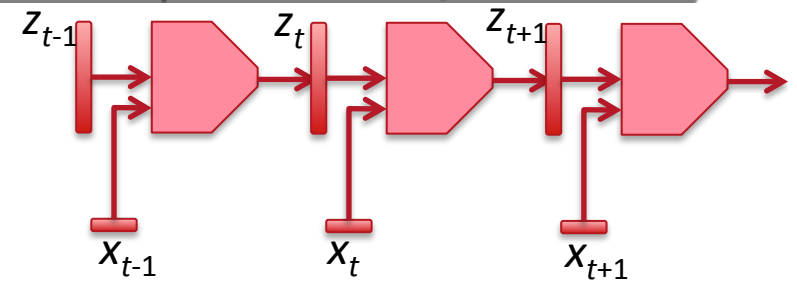
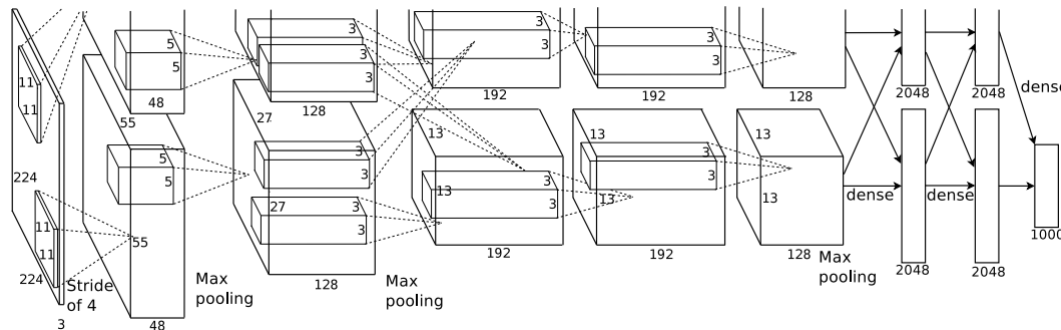
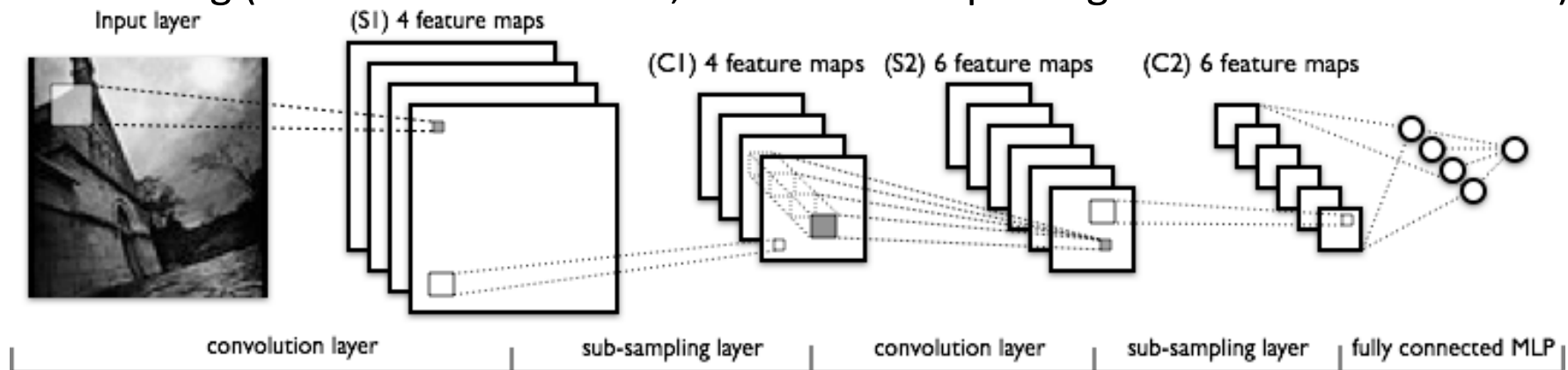
- **Dropouts** trick: during training multiply neuron output by random bit ( $p=0.5$ ), during test by 0.5
- Used in deep supervised networks
- Similar to denoising auto-encoder, but corrupting every layer
- Works better with some non-linearities (rectifiers, maxout)  
(Goodfellow et al. ICML 2013)
- Equivalent to averaging over exponentially many architectures
  - Used by Krizhevsky et al to break through ImageNet SOTA
  - Also improves SOTA on CIFAR-10 (18→16% err)
  - Knowledge-free MNIST with DBMs (.95→.79% err)
  - TIMIT phoneme classification (22.7→19.7% err)

# Dropout Regularizer: Super-Efficient Bagging



# Temporal & Spatial Inputs: Convolutional & Recurrent Nets

- Local connectivity across time/space
- Sharing weights across time/space (translation equivariance)
- Pooling (translation invariance, cross-channel pooling for learned invariances)

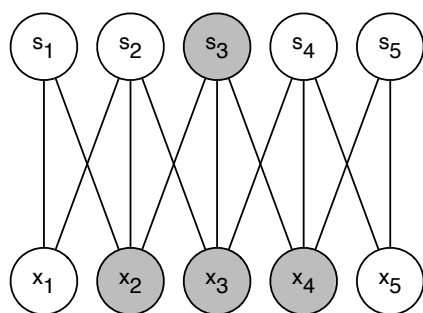


Recurrent nets (RNNs) can summarize information from the past

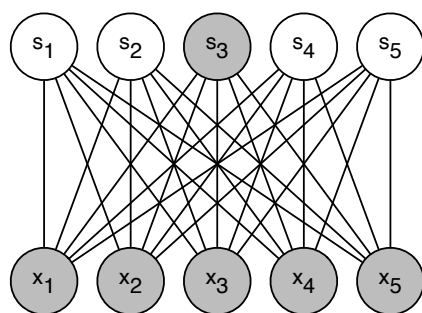
Bidirectional RNNs also summarize information from the future

Convolution = sparse connectivity + parameter sharing

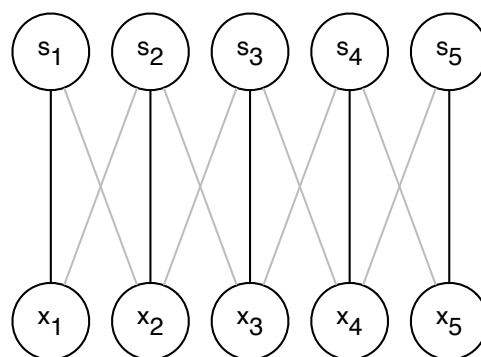
$$s[t] = (x * w)(t) = \sum_{a=-\infty}^{\infty} x[a]w[t - a]$$



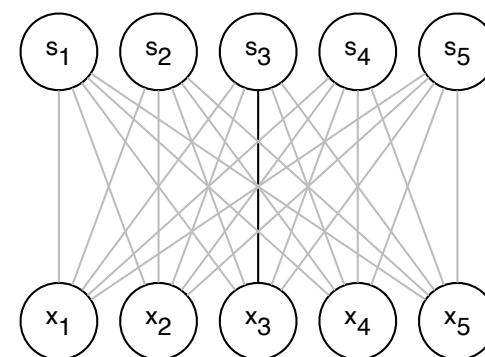
sparse



dense



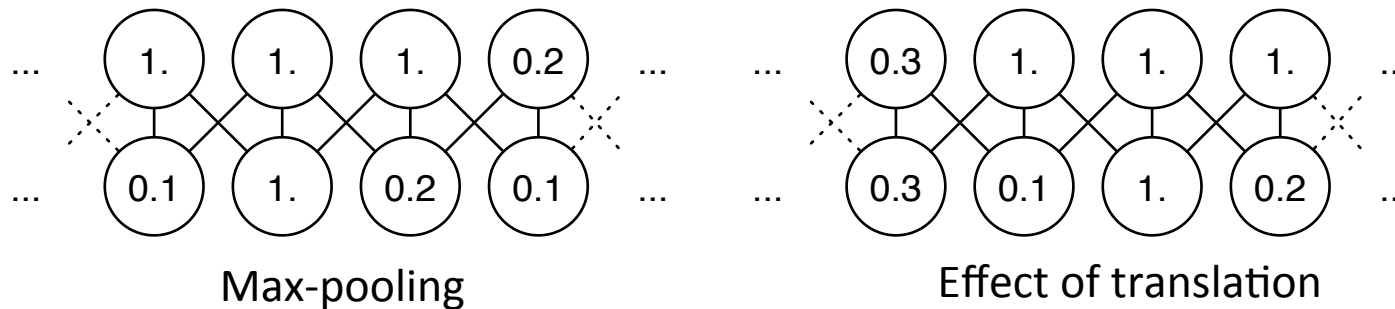
shared



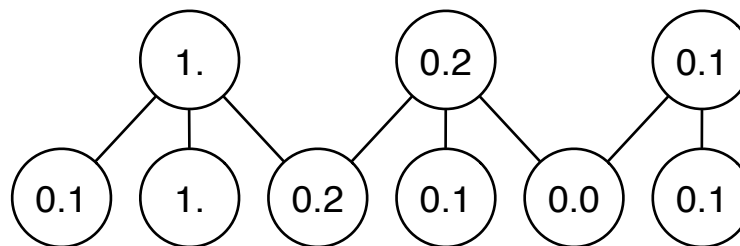
not shared

# Pooling Layers

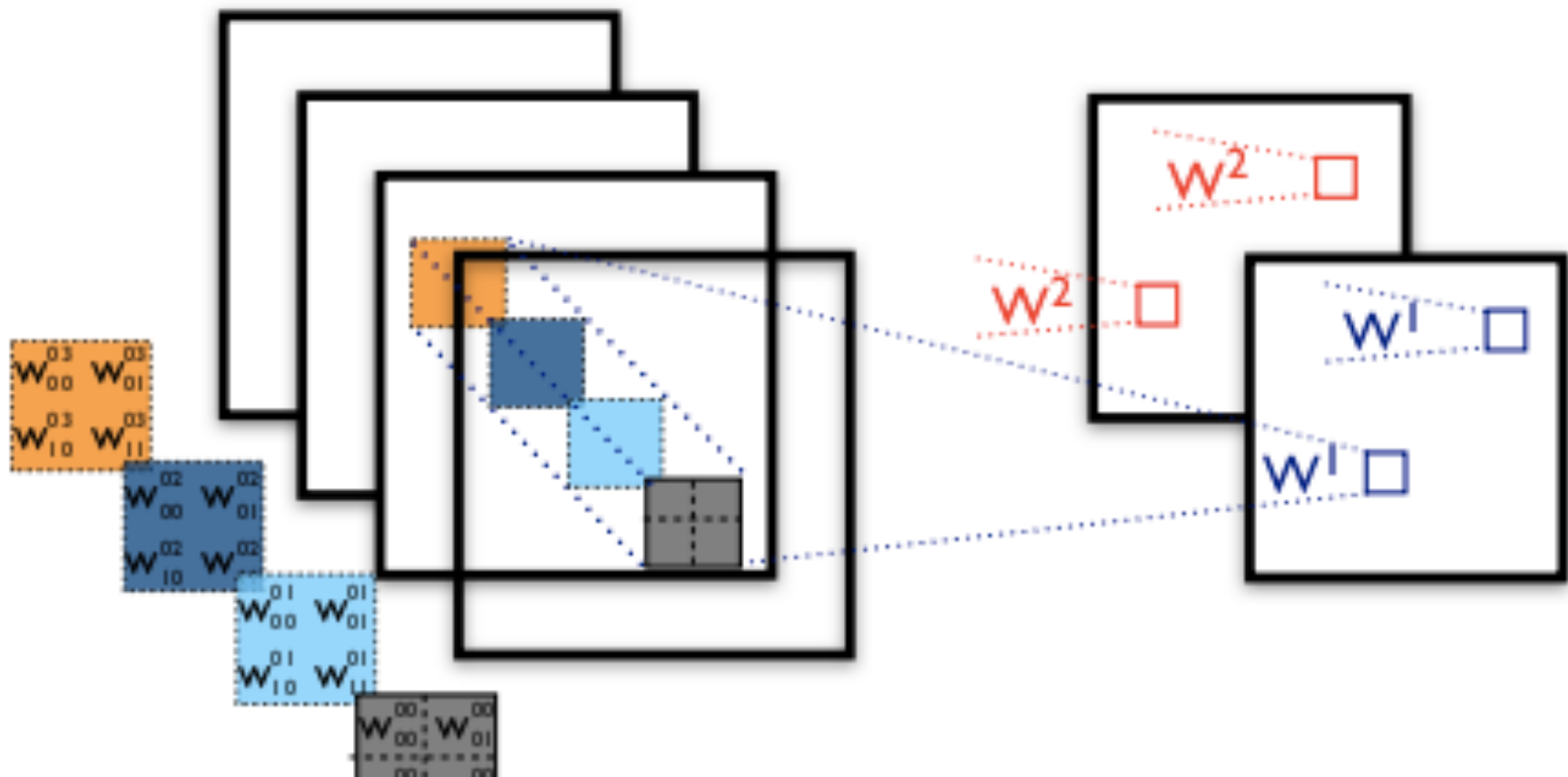
- Aggregate to achieve local invariance



- Subsampling to reduce temporal/spatial scale and computation

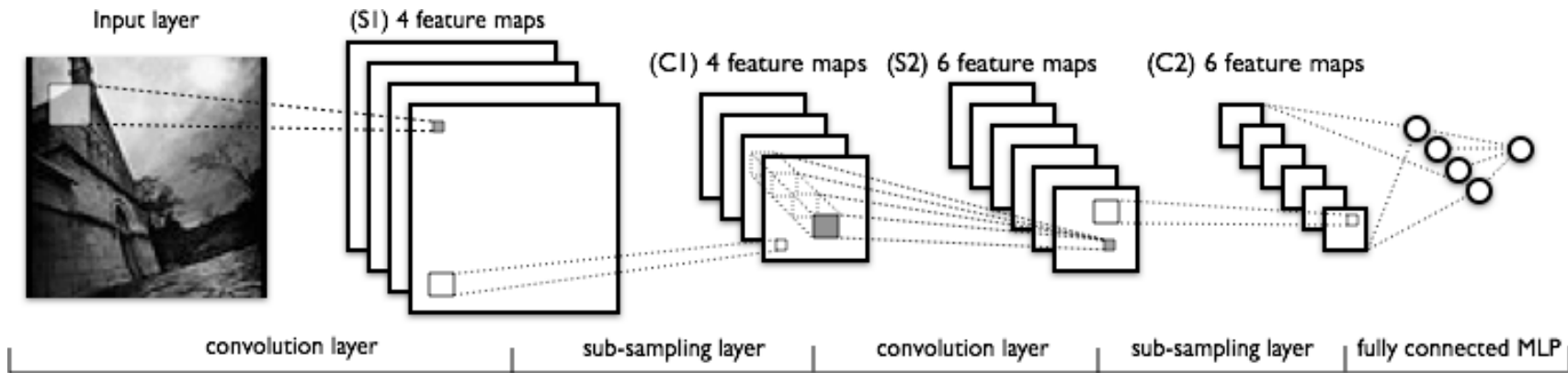


# Multiple Convolutions: Feature Maps



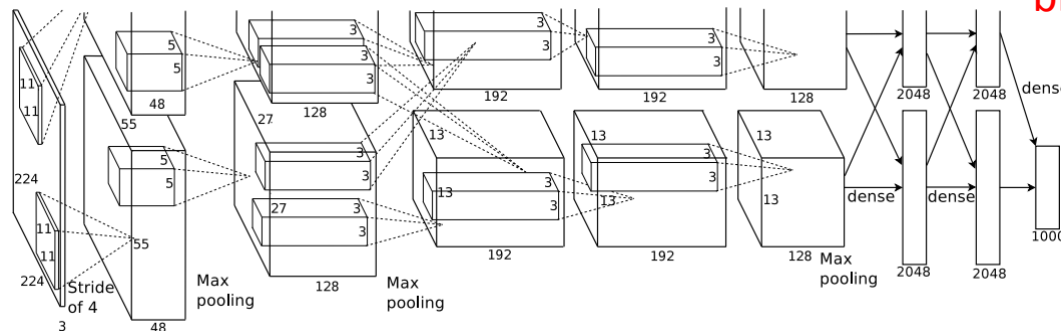
## Alternating convolutions & pooling

- Inspired by visual cortex, idea from Fukushima's Neocognitron, combined with back-prop and developed by **LeCun** since 1989



- Increasing number of features, decreasing spatial resolution
  - Top layers are fully connected
- Krizhevsky, Sutskever & Hinton

Krizhevsky, Sutskever & Hinton 2012  
breakthrough in object recognition

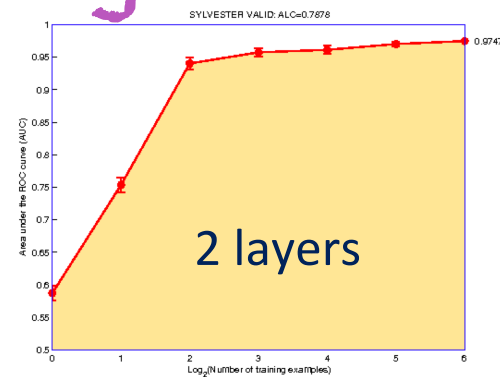
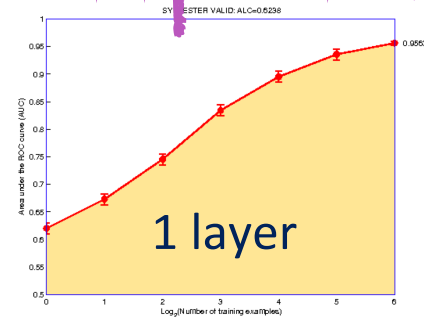
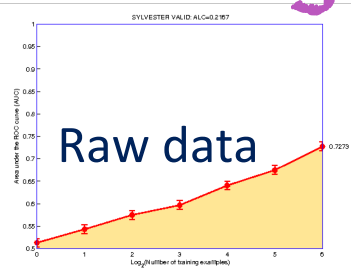


Distributed Representations  
& Neural Nets:

How to do **unsupervised**  
training?

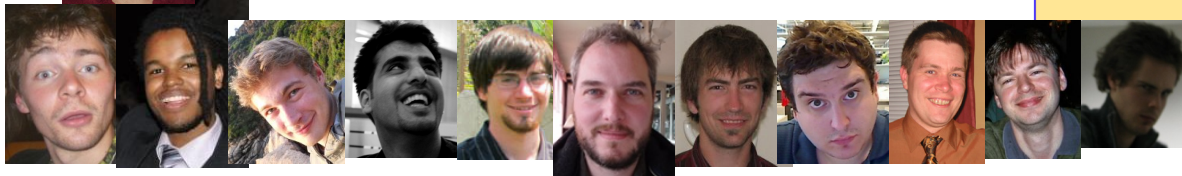
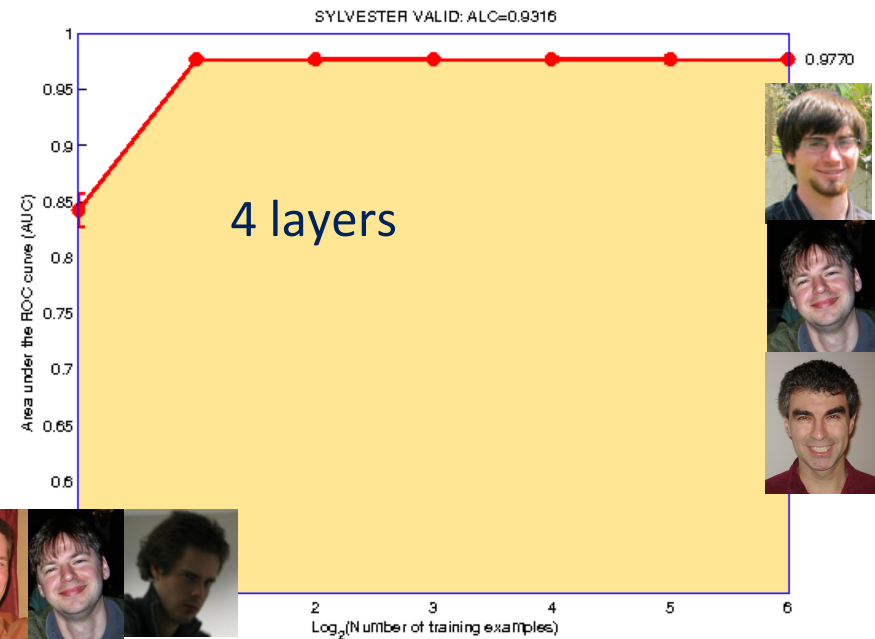
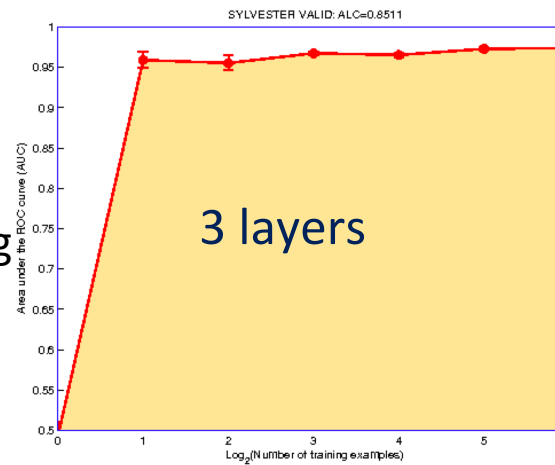


# Unsupervised and Transfer Learning Challenge + Transfer Learning Challenge: Deep Learning 1st Place



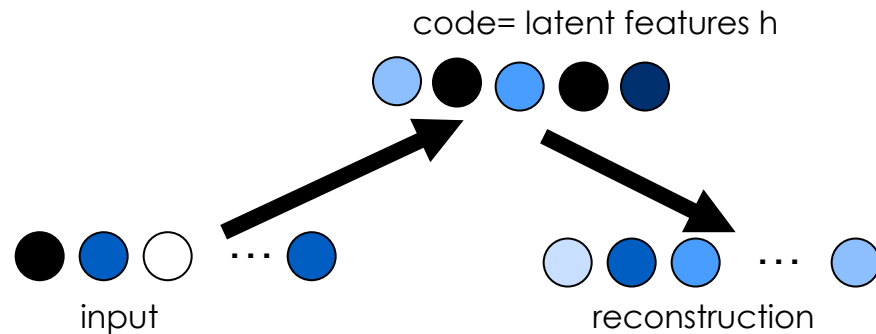
NIPS'2011  
Transfer  
Learning  
Challenge  
Paper:  
ICML'2012

ICML'2011  
workshop on  
Unsup. &  
Transfer Learning



# PCA

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

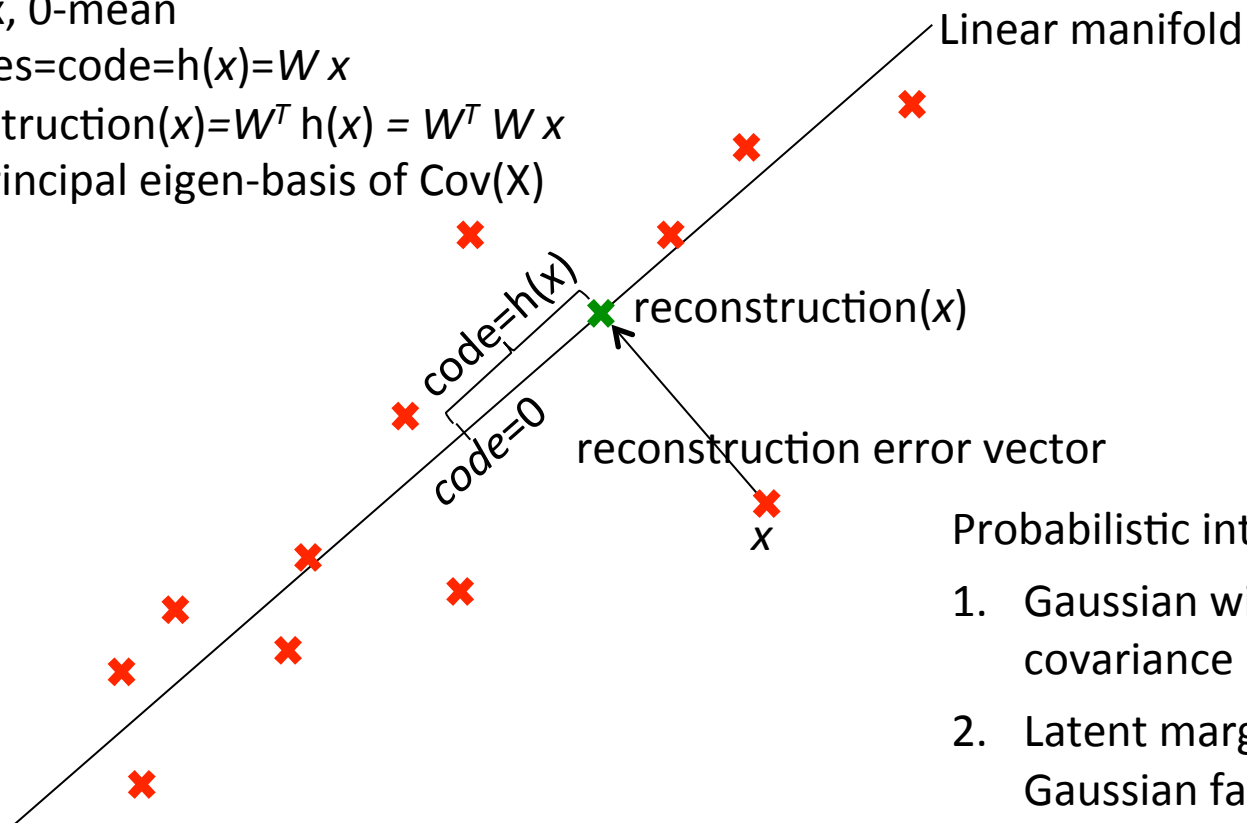


input  $x$ , 0-mean

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

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

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



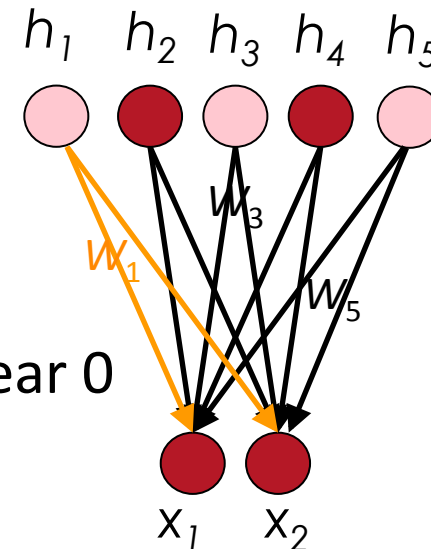
Probabilistic interpretations:

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:

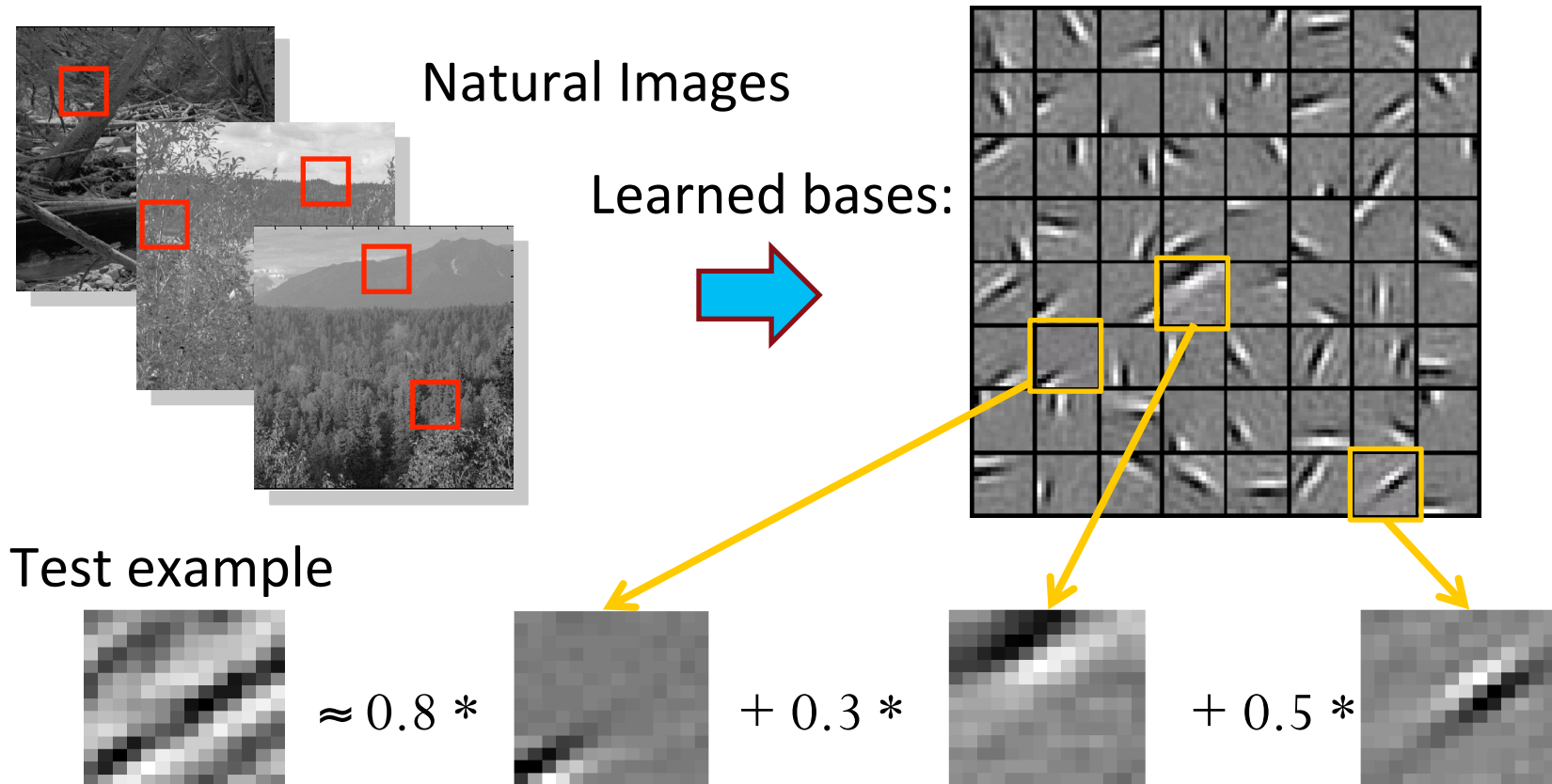
$$P(x, h) = \underbrace{P(h)}_{\text{factors prior}} \underbrace{P(x|h)}_{\text{likelihood}}$$

- $P(h)$  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  $x | h$  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  $h_i$



$$\begin{array}{|c|} \hline x \\ \hline \end{array} = .9 \times \begin{array}{|c|} \hline W_1 \\ \hline \end{array} + .8 \times \begin{array}{|c|} \hline W_3 \\ \hline \end{array} + .7 \times \begin{array}{|c|} \hline W_5 \\ \hline \end{array}$$

# Sparse autoencoder illustration for images

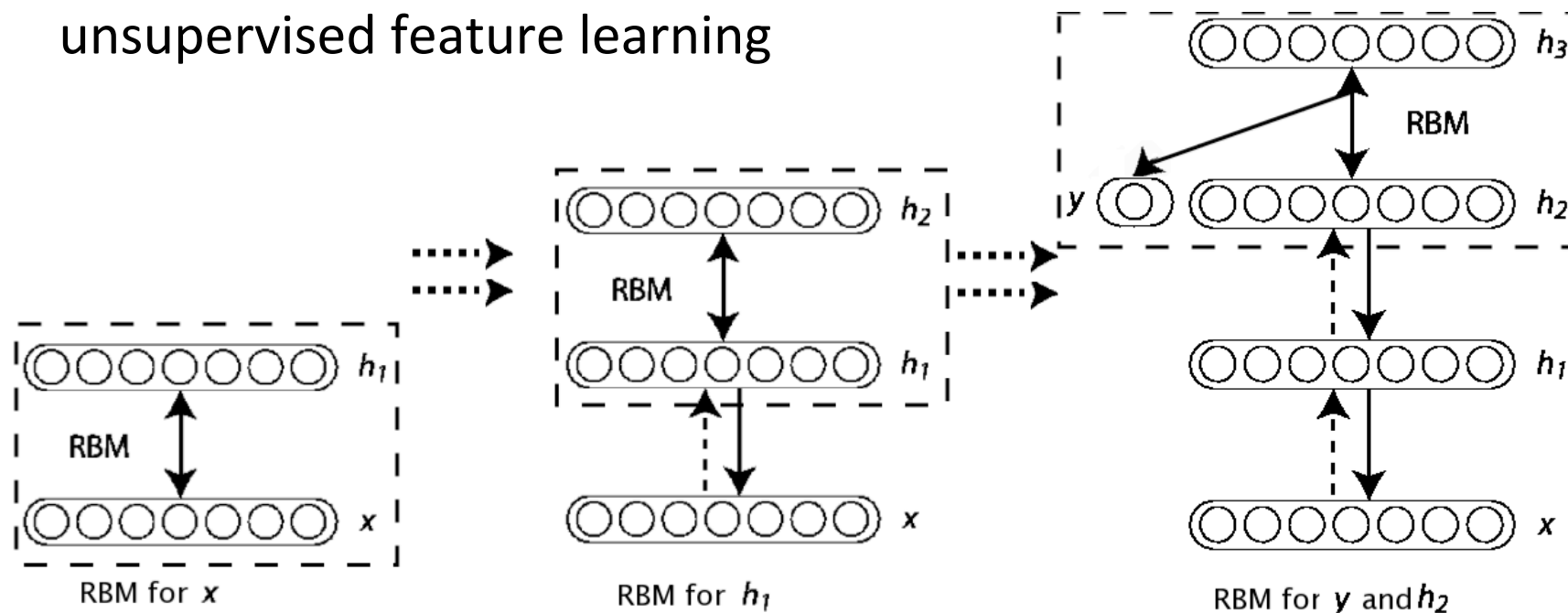


$$[h_1, \dots, h_{64}] = [0, 0, \dots, 0, \mathbf{0.8}, 0, \dots, 0, \mathbf{0.3}, 0, \dots, 0, \mathbf{0.5}, 0]$$

(feature representation)

# Stacking Single-Layer Learners

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



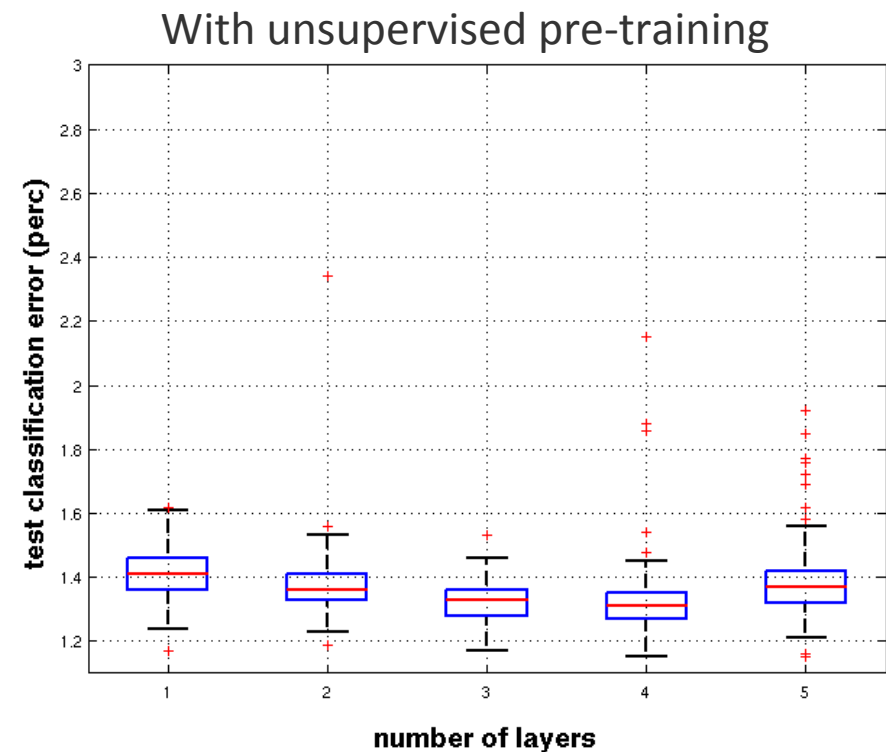
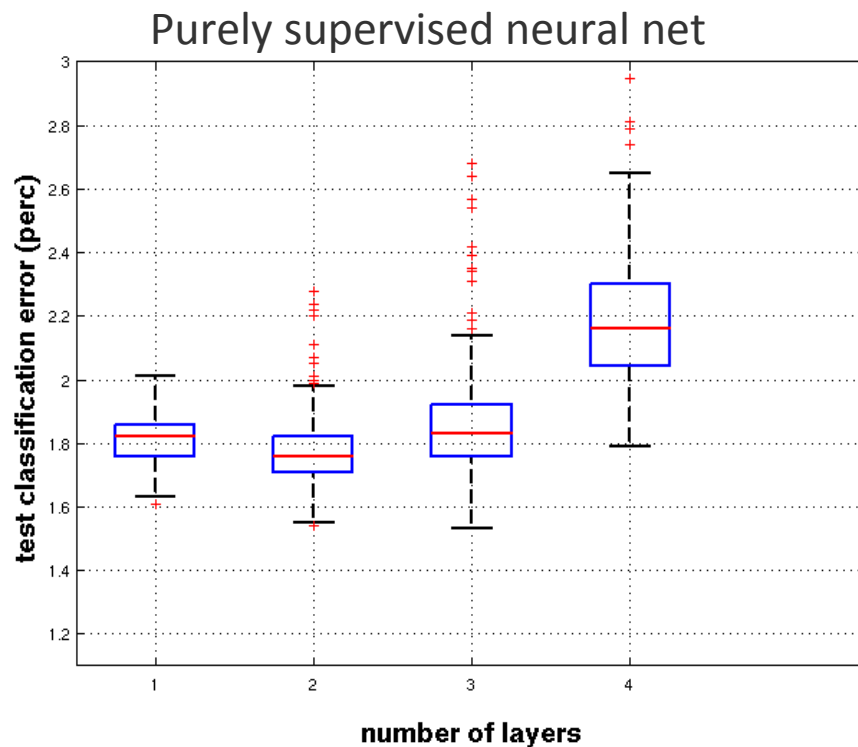
Stacking Restricted Boltzmann Machines (RBM)  $\rightarrow$  Deep Belief Network (DBN)

# Effective deep Learning first became possible with unsupervised pre-training

[Erhan et al., JMLR 2010]



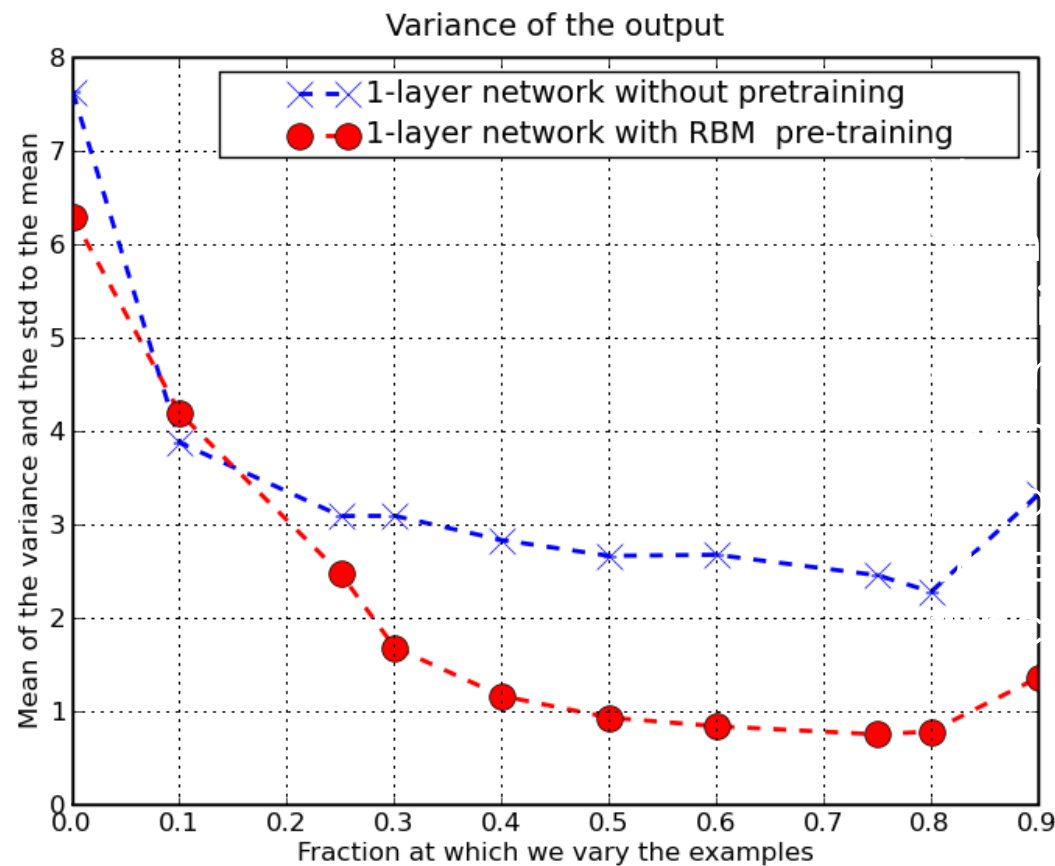
(with RBMs and Denoising Auto-Encoders)



# Optimizing Deep Non-Linear Composition of Functions Seems Hard

- Failure of training deep supervised nets before 2006
- Regularization effect + optimization effect of unsupervised pre-training
- Is optimization difficulty due to
  - ill-conditioning?
  - local minima?
  - something else?
- The jury is still out, but we now have success stories of training deep supervised nets without unsupervised pre-training

# Initial Examples Matter More (critical period?)



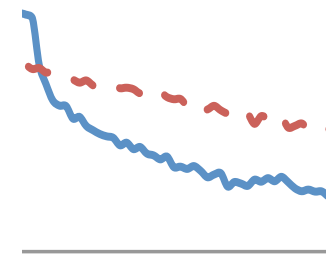


# Order & Selection of Examples Matters

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



- Curriculum learning
  - (Bengio et al 2009, Krueger & Dayan 2009)
  - **Start with easier examples**
- Faster convergence to a better local minimum in deep architectures



— curriculum  
- - no-curriculum

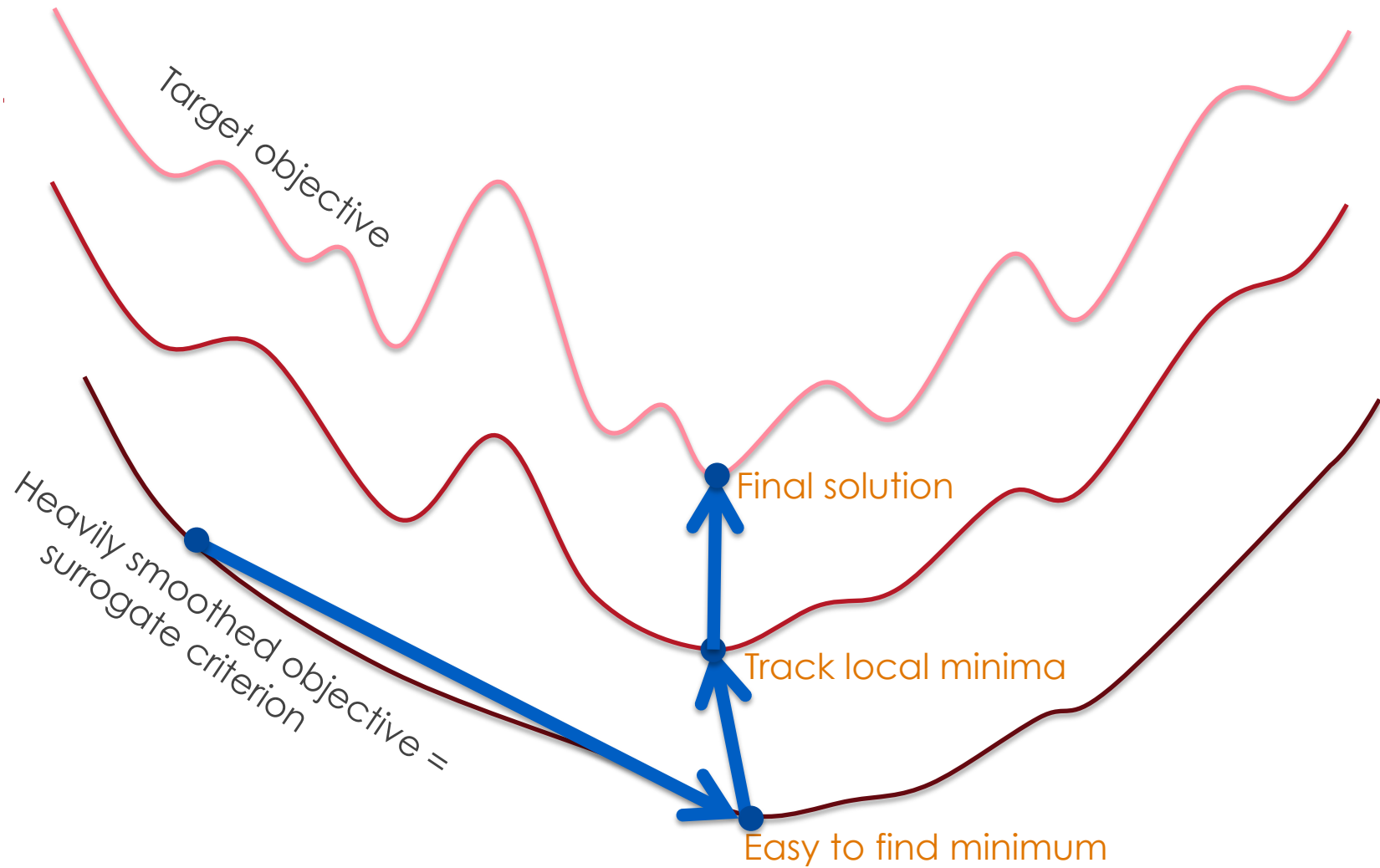
# Curriculum Learning

Guided learning helps training humans and animals



Start from simpler examples / easier tasks (Piaget 1952, Skinner 1958)

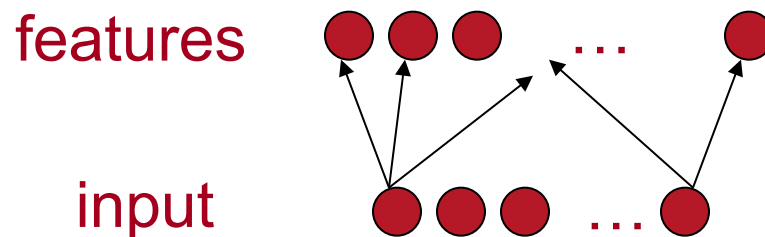
# Continuation Methods



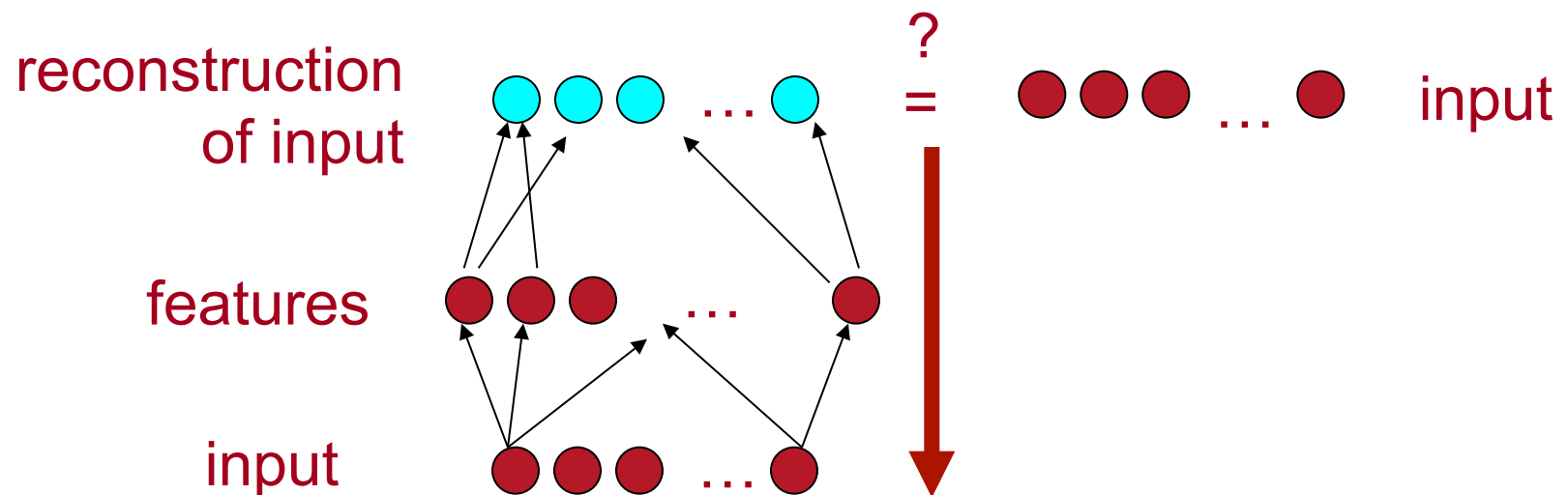
# Layer-wise Unsupervised Learning

input      ● ● ● ... ●

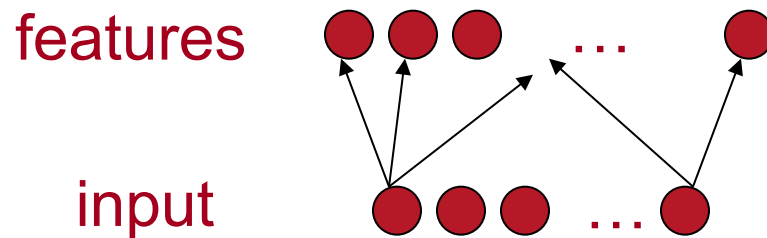
# Layer-Wise Unsupervised Pre-training



# Layer-Wise Unsupervised Pre-training



# Layer-Wise Unsupervised Pre-training

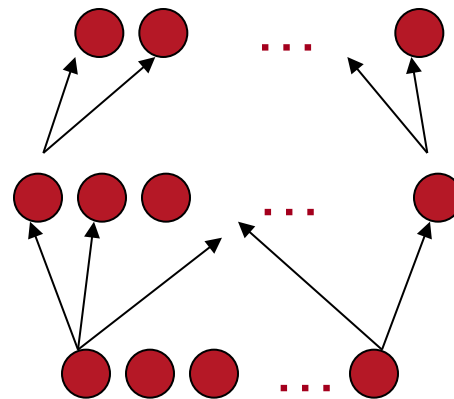


# Layer-Wise Unsupervised Pre-training

More abstract  
features

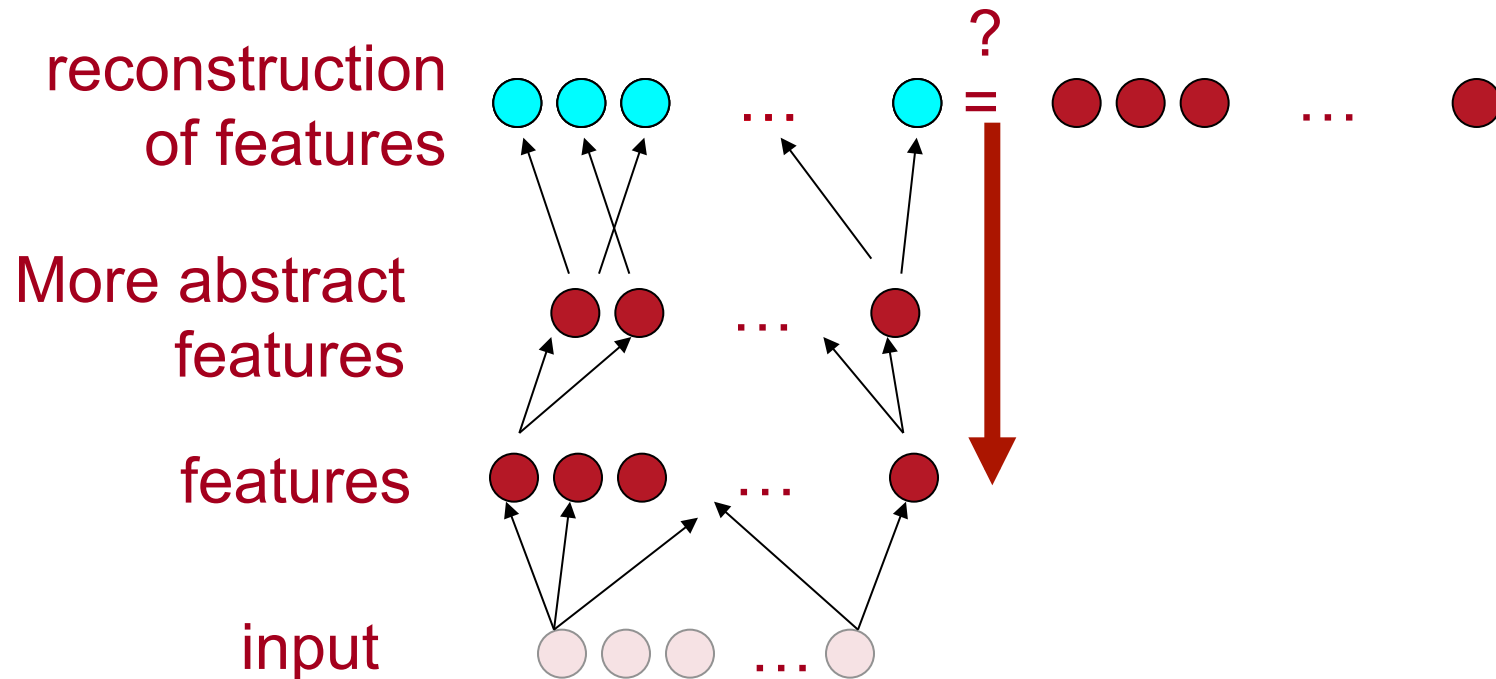
features

input





# Layer-wise Unsupervised Learning

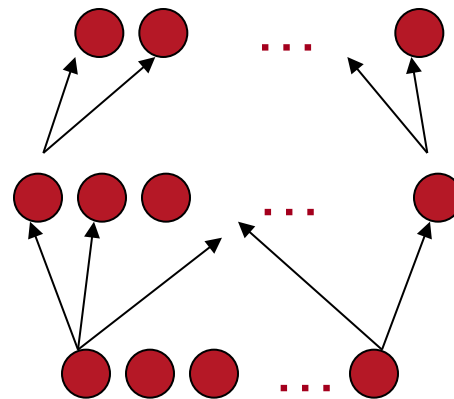


# Layer-Wise Unsupervised Pre-training

More abstract  
features

features

input



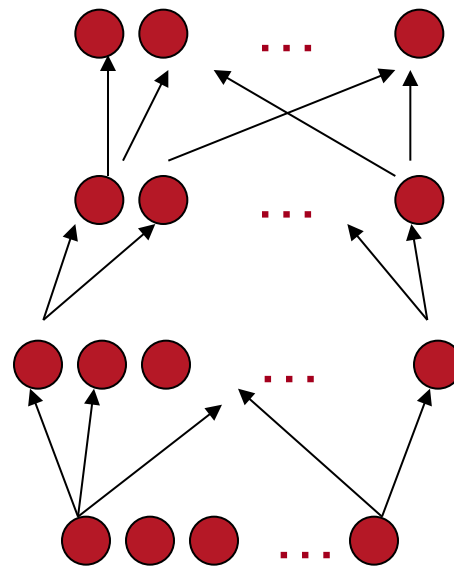
# Layer-wise Unsupervised Learning

Even more abstract  
features

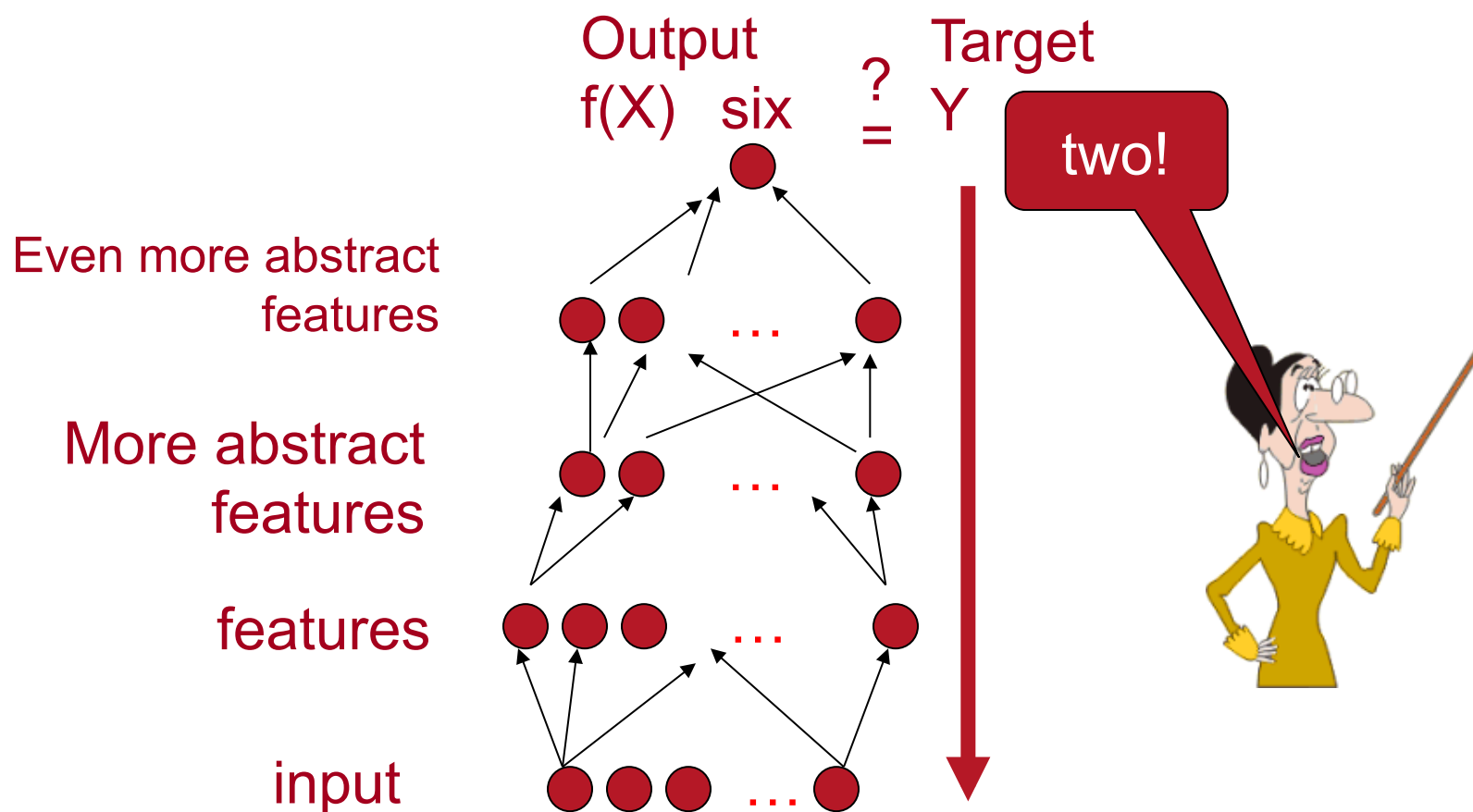
More abstract  
features

features

input

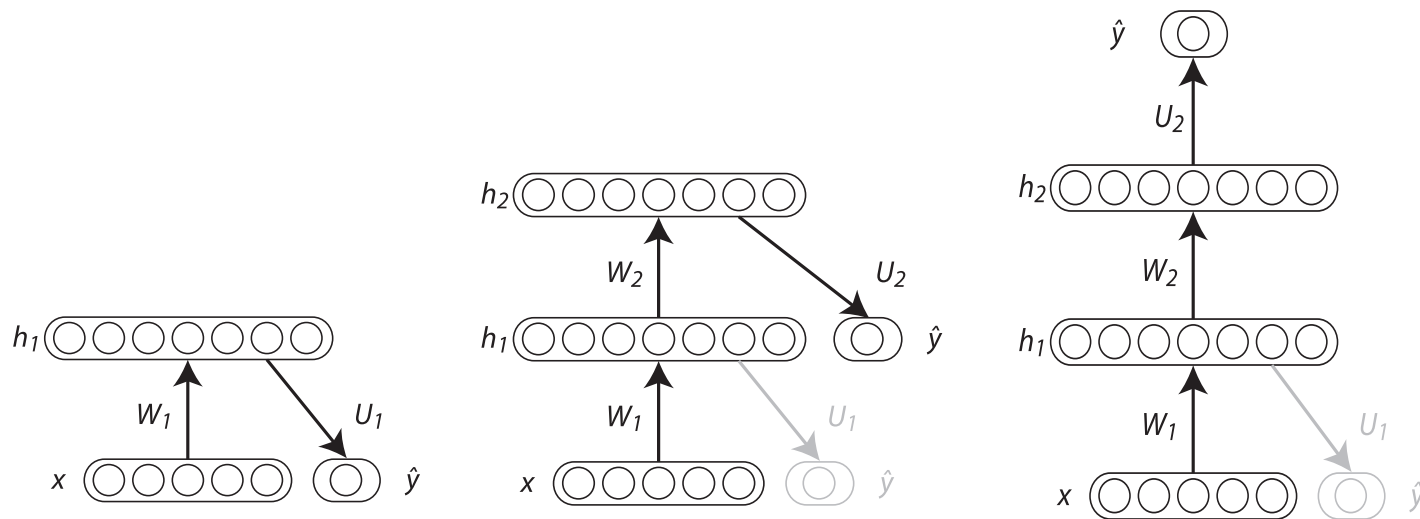


# Supervised Fine-Tuning



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

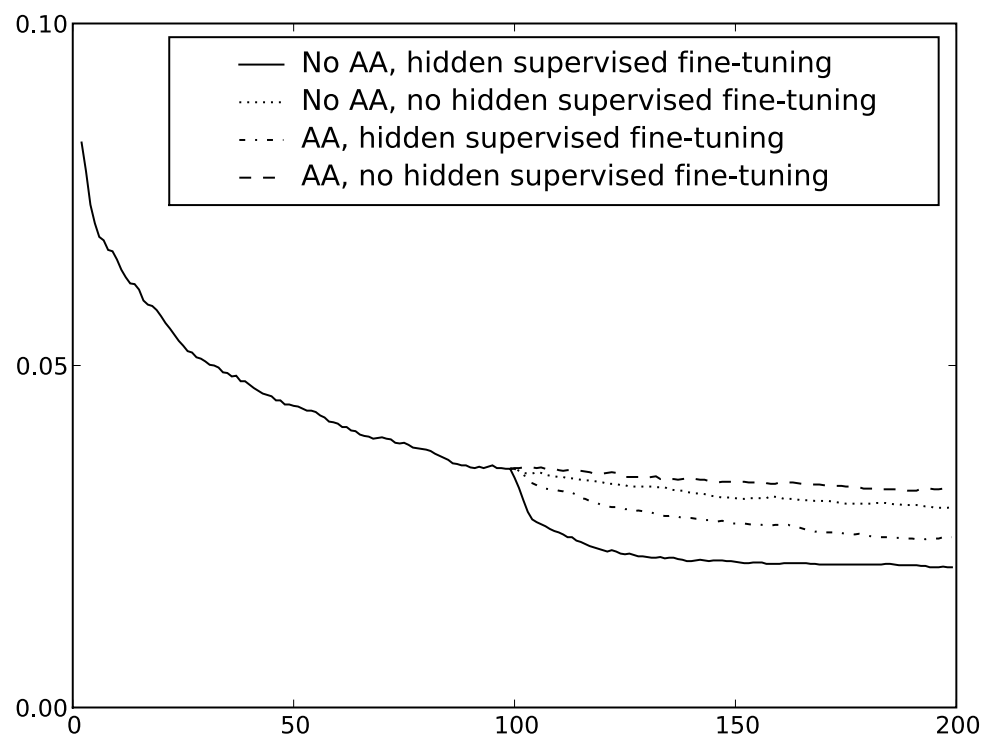
# Greedy Layerwise Supervised Training



Generally worse than unsupervised pre-training but better than ordinary training of a deep neural network (Bengio et al. NIPS'2006). Has been used successfully on large labeled datasets, where unsupervised pre-training did not make as much of an impact.

# 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



# Understanding the difficulty of training deep feedforward supervised neural networks

(Glorot & Bengio, AISTATS 2010)



## Study the activations and gradients

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

*First demonstration that deep supervised nets can be successfully trained almost as well as with unsupervised pre-training, by setting up the optimization problem appropriately...*

# Restricted Boltzmann Machines

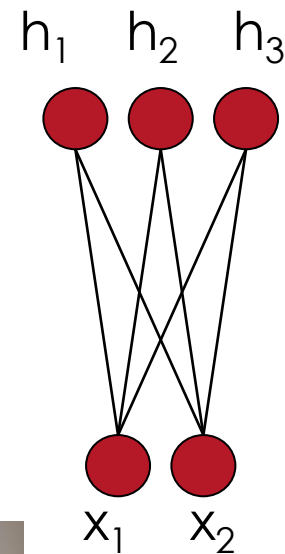


# Undirected Models: the Restricted Boltzmann Machine

[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 into product of  $P(h_i | x)$
- See Bengio (2009) detailed monograph/review: *“Learning Deep Architectures for AI”*.
- See Hinton (2010) *“A practical guide to training Restricted Boltzmann Machines”*



# Boltzmann Machines & MRFs

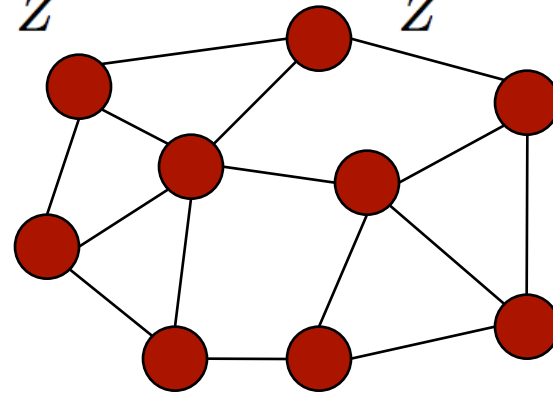
- Boltzmann machines:

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

- Markov Random Fields:

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

Soft constraint / probabilistic statement



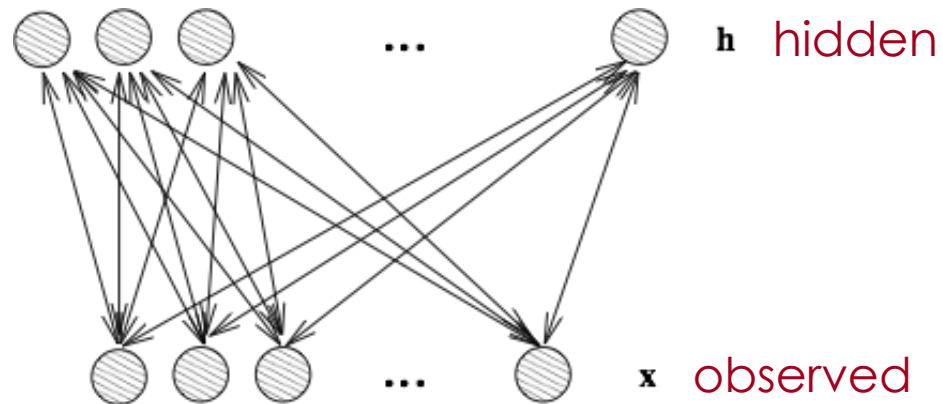
Undirected  
graphical  
models

- More interesting with latent variables!

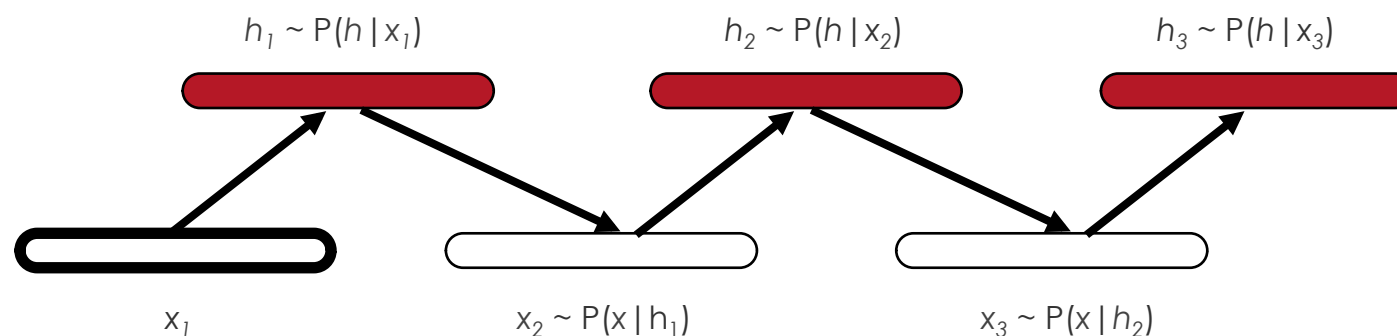
# Restricted Boltzmann Machine (RBM)

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

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



# Block Gibbs Sampling in RBMs



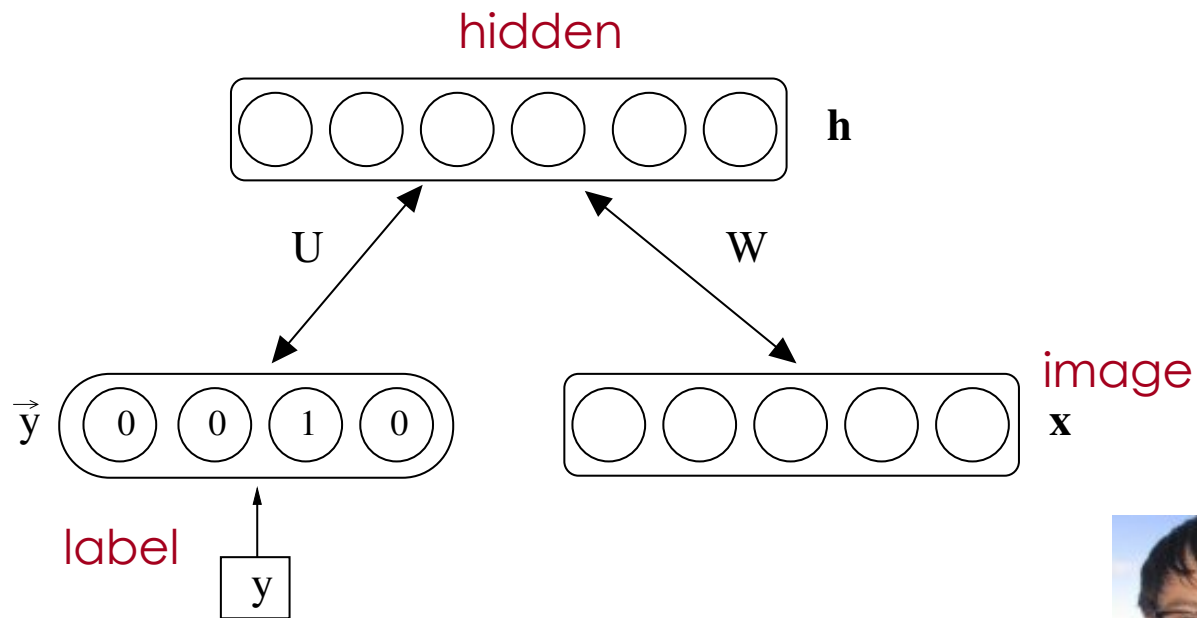
$P(h | x)$  and  $P(x | h)$  factorize

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

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

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

# 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

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

# 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  $Z$

# 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

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

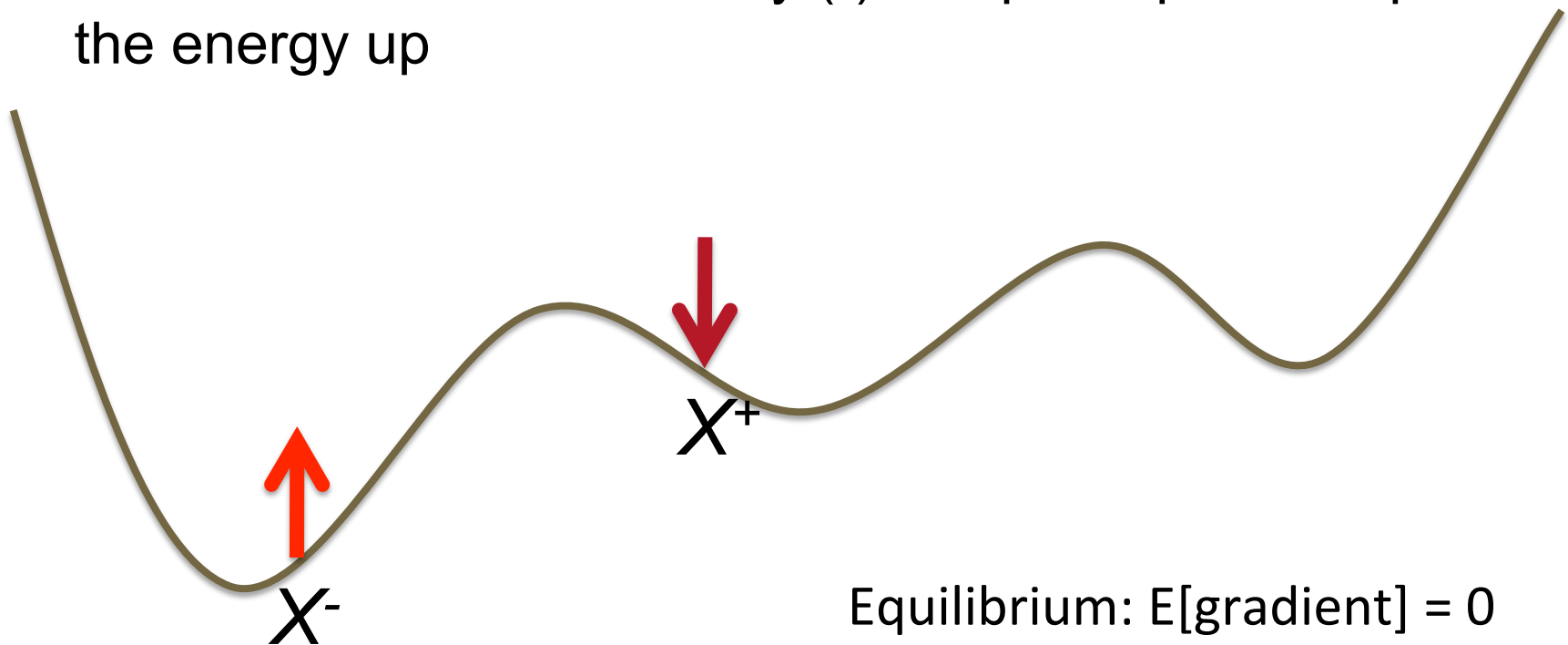
- Gradient has two components:

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

- 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

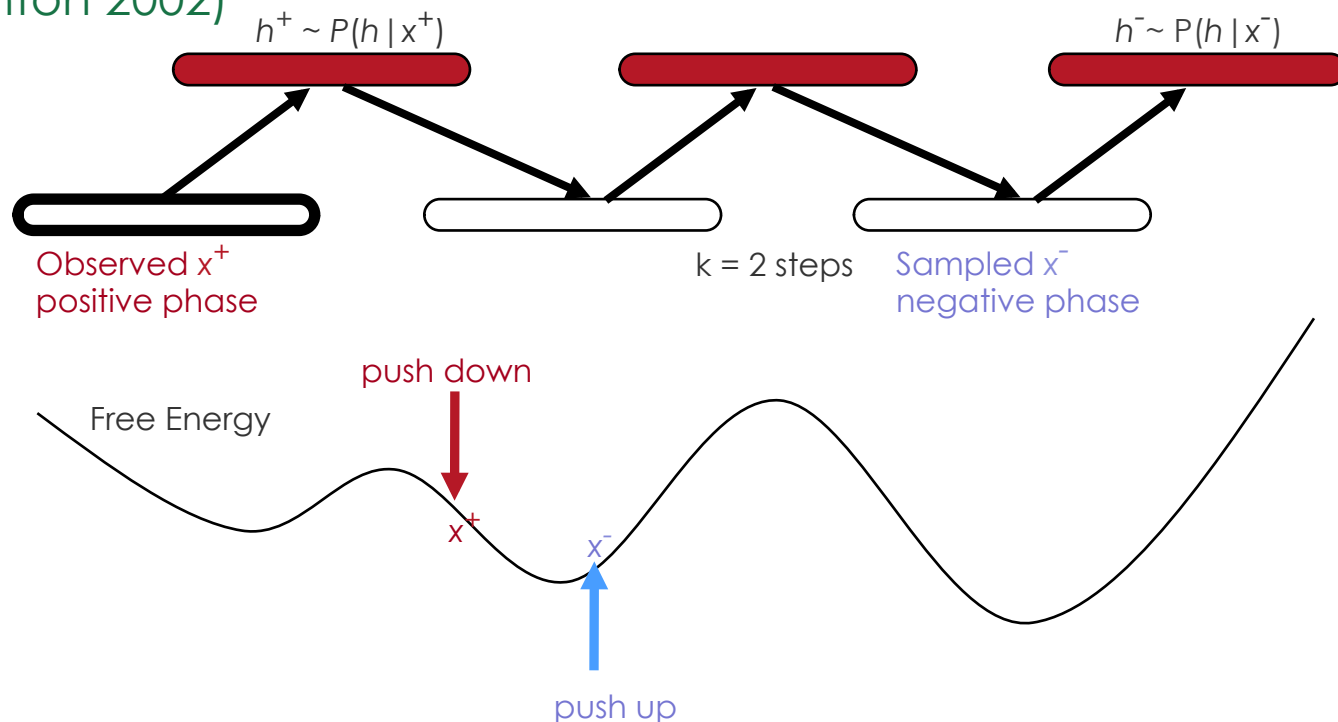
**Tempered MCMC:** use higher temperature to escape modes

# Contrastive Divergence



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

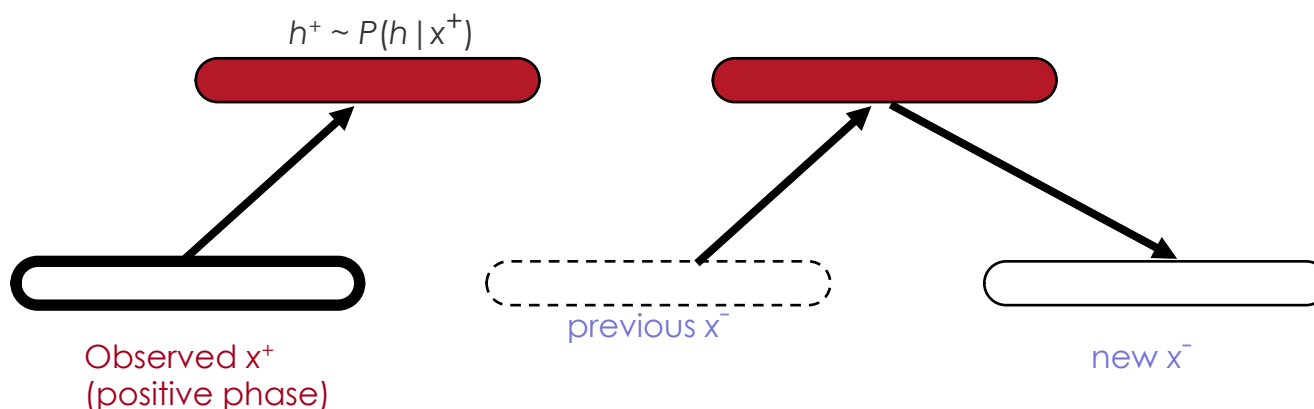
(Hinton 2002)



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

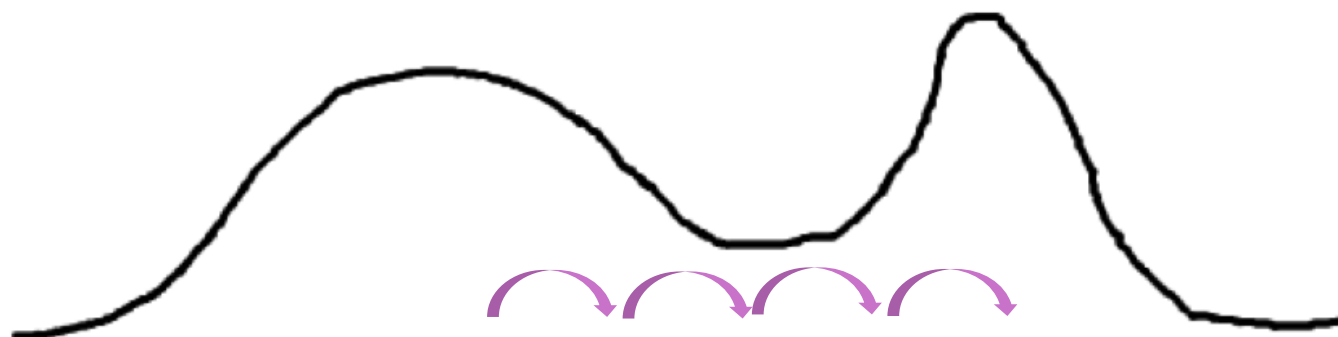
Run negative Gibbs chain in background while weights slowly change (Younes 1999, Tieleman 2008):

- Guarantees (Younes 1999; Yuille 2005)
- If learning rate decreases in  $1/t$ , chain mixes before parameters change too much, chain stays converged when parameters change

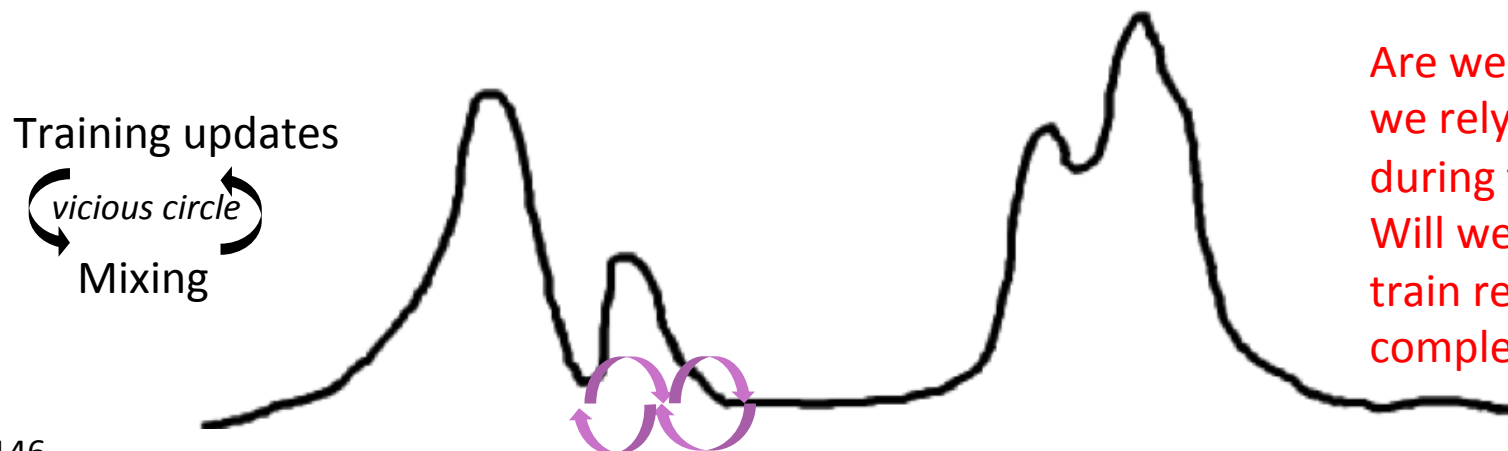


# Obstacle: Vicious Circle Between Learning and MCMC Sampling

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



- Later on, hard to cross empty voids between modes



Are we doomed if we rely on MCMC during training?  
Will we be able to train really large & complex models?

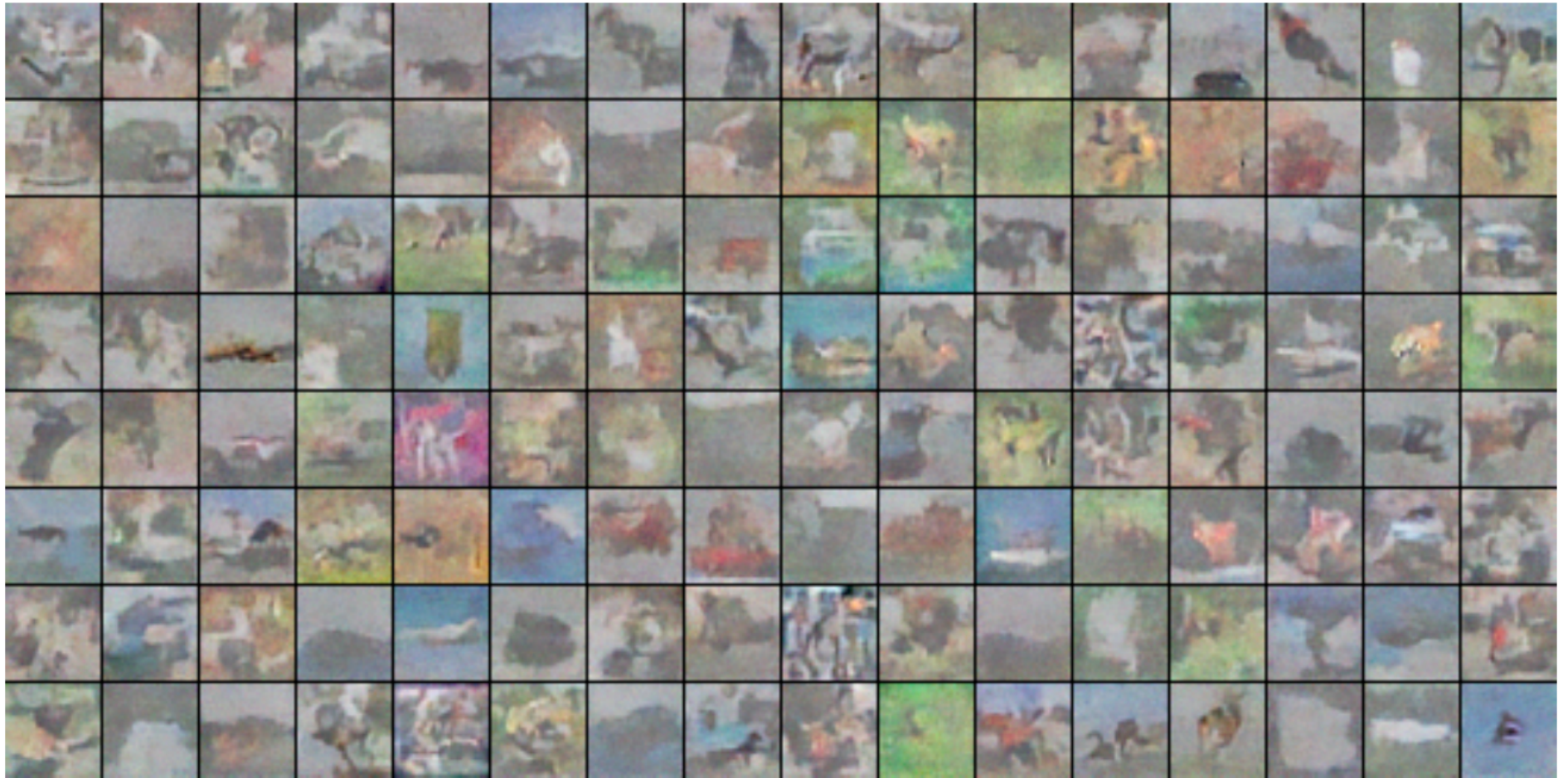


## Some RBM Variants

- 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



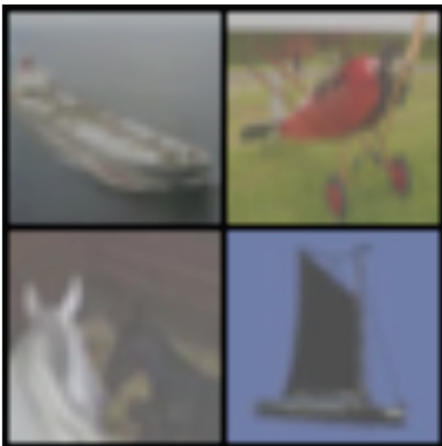


# ssRBM is not Cheating

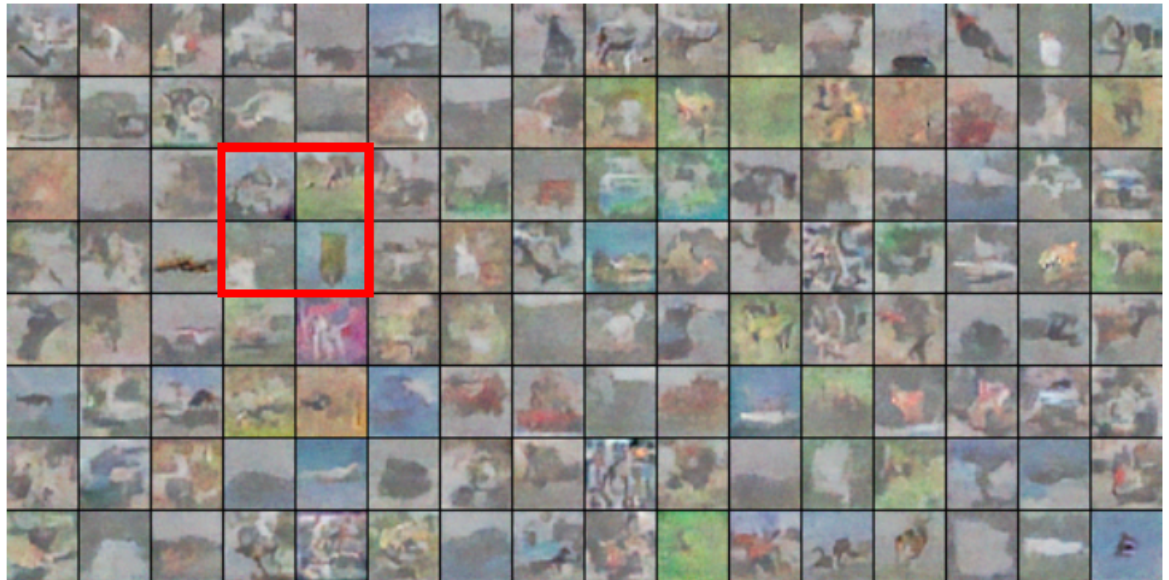
Samples from  $\mu$ -ssRBM:



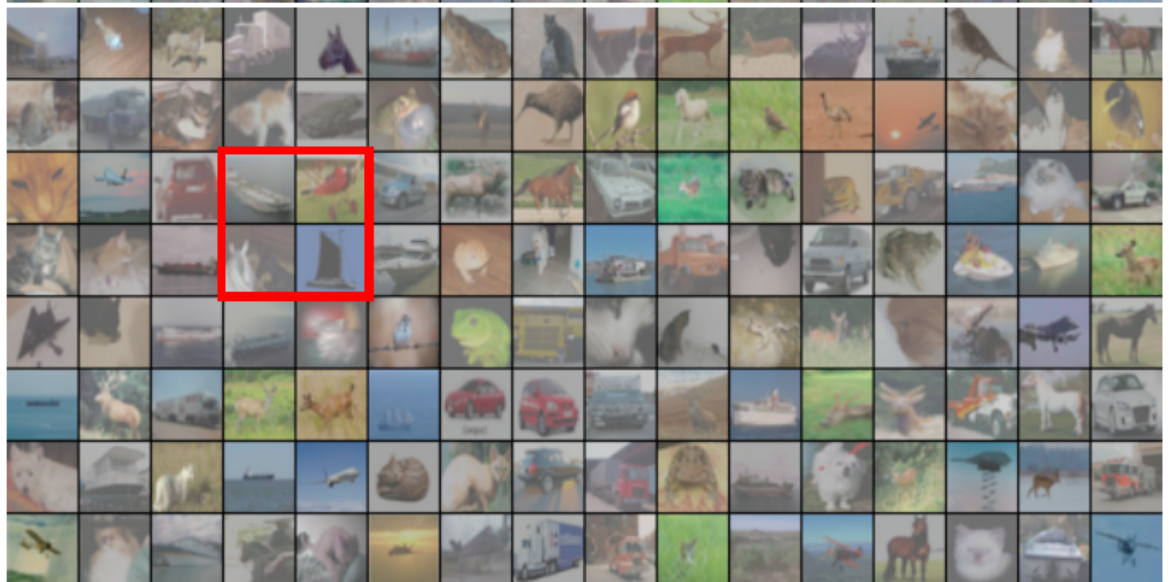
Nearest examples in CIFAR:  
(least square dist.)



Generated samples

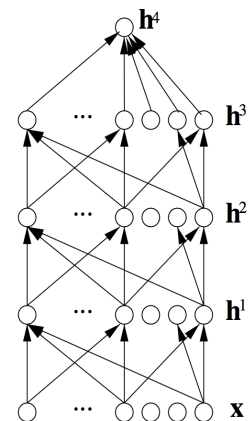


Training examples



# Auto-Encoders & Variants: Learning a computational graph

# Computational Graphs



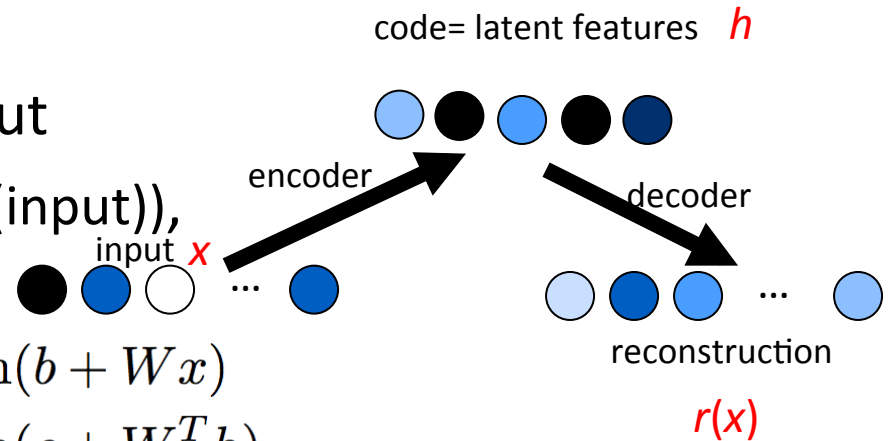
- Operations for particular task
- Neural nets' structure = computational graph for  $P(y|\mathbf{x})$
- Graphical model's structure  $\neq$  computational graph for inference
- Recurrent nets & graphical models

➔ family of computational graphs sharing parameters

- *Could we have a parametrized family of computational graphs defining “the model”?*

# Simple Auto-Encoders

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



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

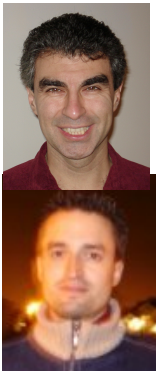
$$\text{reconstruction} = \tanh(c + W^T h)$$

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

- With bottleneck, code = new coordinate system
- Encoder and decoder can have 1 or more layers
- Training deep auto-encoders notoriously difficult

# Link Between Contrastive Divergence and Auto-Encoder Reconstruction Error Gradient

- (Bengio & Delalleau 2009):



- CD-2k estimates the log-likelihood gradient from 2k diminishing terms of an expansion that mimics the Gibbs steps
- reconstruction error gradient looks only at the first step, i.e., is a kind of mean-field approximation of CD-0.5

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

# I finally understand what auto-encoders do!

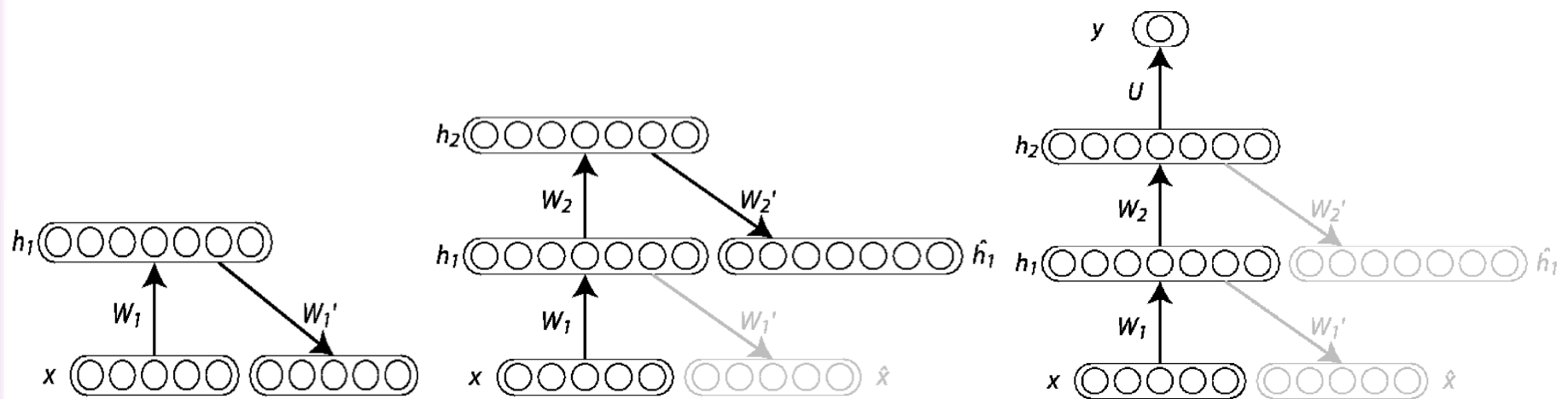
- Try to carve holes in  $\|r(x)-x\|^2$  or  $-\log P(x | h(x))$  at examples



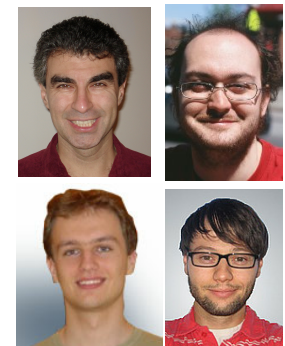
- Vector  $r(x)-x$  points in direction of increasing prob., i.e. estimate  $\text{score} = d \log p(x) / dx$ : learn **score** vector field = **local mean**
- Generalize (*valleys*) in between above holes to form *manifolds*
  - $dr(x)/dx$  estimates the **local covariance** and is linked to the Hessian  $d^2 \log p(x) / dx^2$
- **A Markov Chain associated with AEs estimates the data-generating distribution (Bengio et al, arxiv 1305.663, 2013)**



# 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) Reconstruction Loss

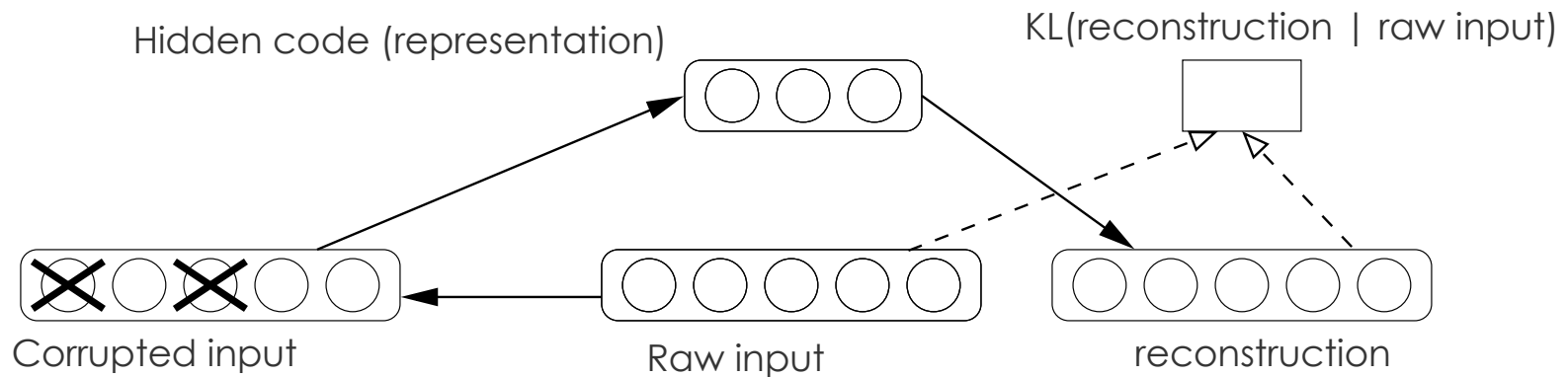
- Discrete inputs: cross-entropy for binary inputs
  - $-\sum_i x_i \log r_i(x) + (1-x_i) \log(1-r_i(x))$  (with  $0 < r_i(x) < 1$ )or log-likelihood reconstruction criterion, e.g., for a multinomial (one-hot) input
  - $-\sum_i x_i \log r_i(x)$  (where  $\sum_i r_i(x) = 1$ , summing over subset of inputs associated with this multinomial variable)
- In general: consider what are appropriate loss functions to predict each of the input variables,  
typically, **reconstruction neg. log-likelihood  $-\log P(x|h(x))$**

# Denoising Auto-Encoder

(Vincent et al 2008)



- Corrupt the input during training only
- Train to 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 pointing towards higher probability direction (Alain & Bengio 2013)



$$r(x) - x \propto d \log p(x) / dx$$

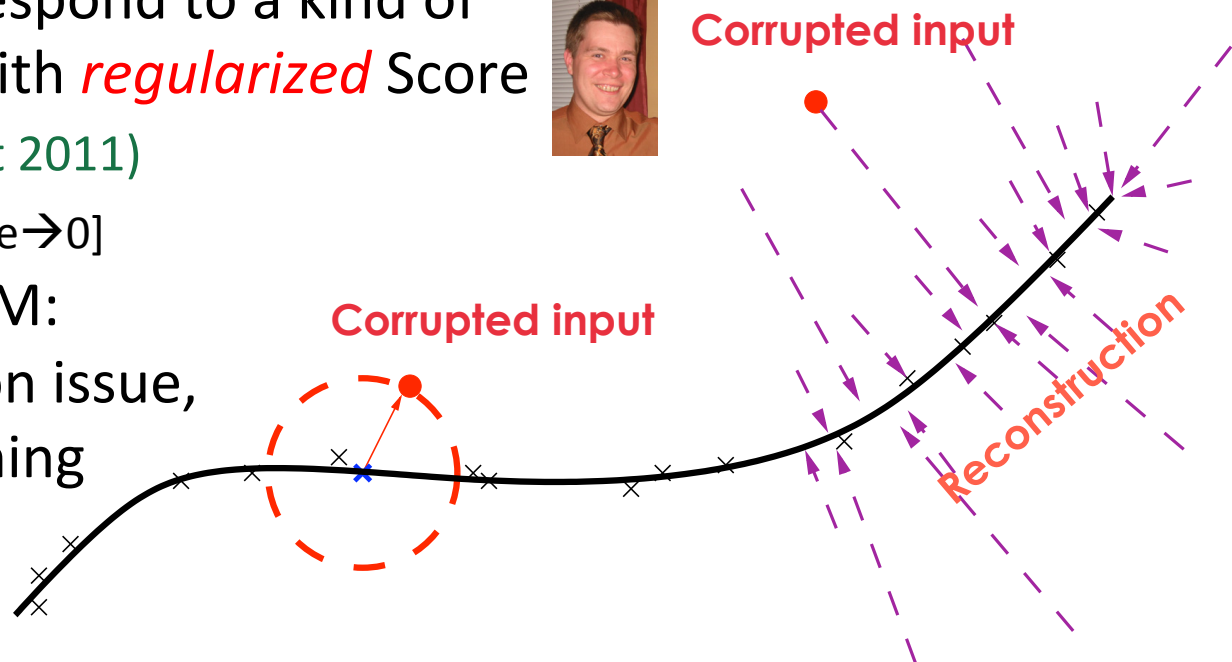
- Some DAEs correspond to a kind of Gaussian RBM with *regularized* Score Matching (Vincent 2011)



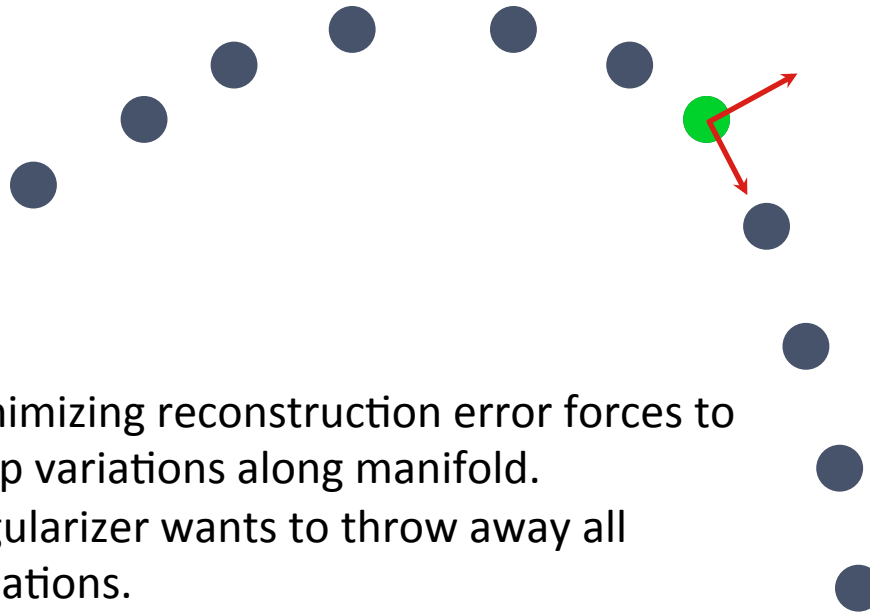
[equivalent when noise  $\rightarrow 0$ ]

- Compared to RBM:  
No partition function issue,  
+ can measure training  
criterion

prior: examples  
concentrate near a  
lower dimensional  
“manifold”



# 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.

# First Theoretical Results on Probabilistic Interpretation of Auto-Encoders

(Vincent 2011, Alain & Bengio 2013)

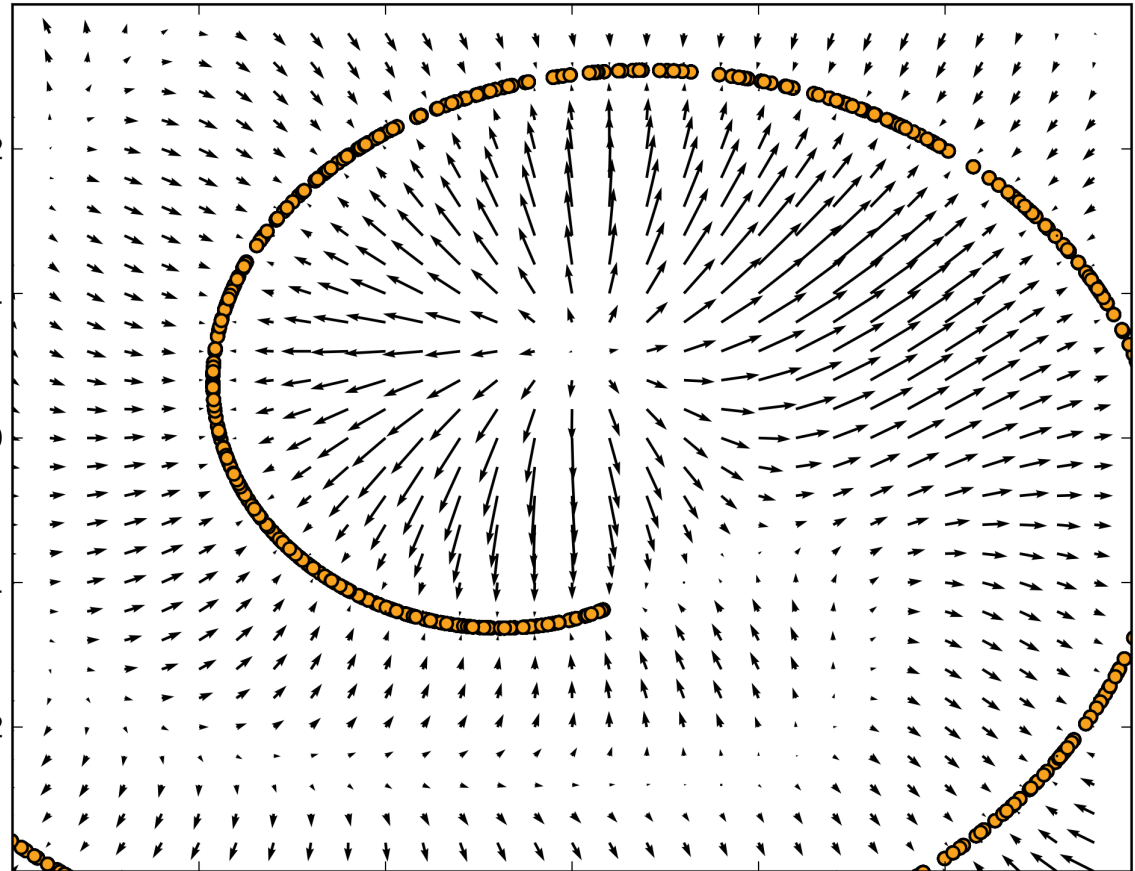
- Continuous  $X$
- Gaussian corruption
- Noise  $\sigma \rightarrow 0$
- Squared reconstruction error  $\|r(X+\text{noise})-X\|^2$

$(r(X)-X)/\sigma^2$  estimates the score  $d \log p(X) / dX$

- Langevin + Metropolis-Hastings can be used to approximately sample from such a model, but mixing was poor

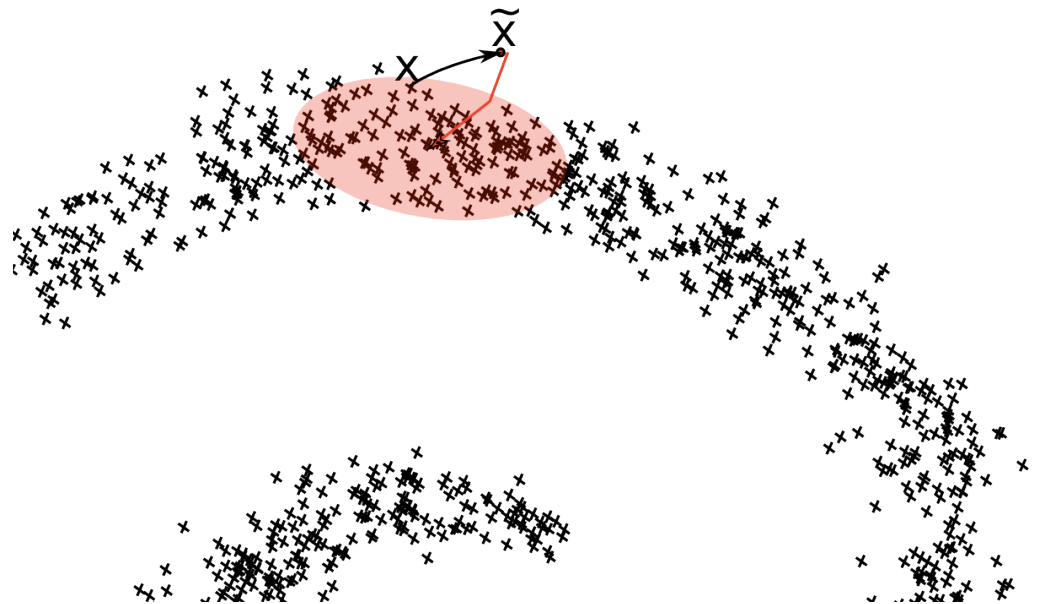
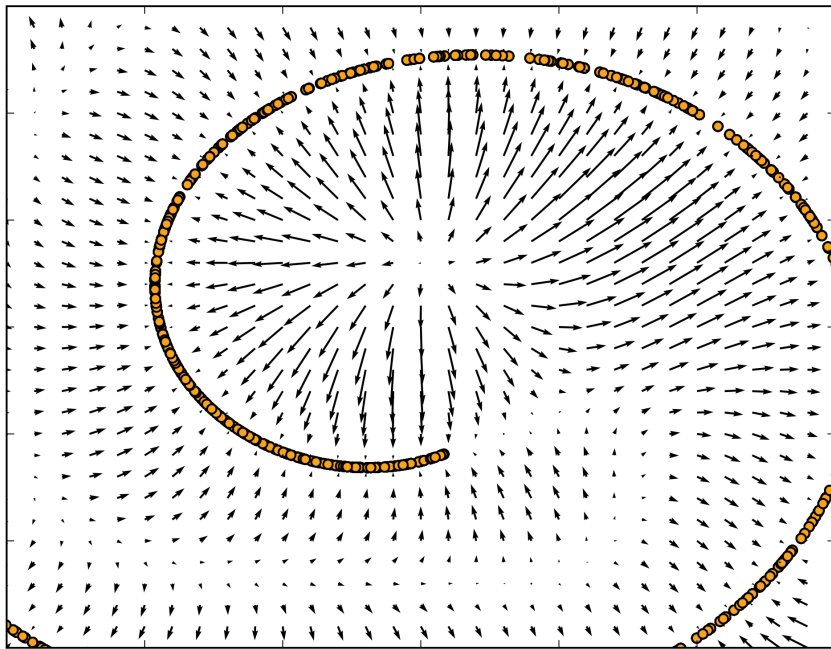
# Learning a Vector Field that Estimates a Gradient Field

- Continuous inputs
- Gaussian corruption
- Squared error
- $\text{Reconstruction}(x) - x$  estimates  $d \log p(x) / dx$
- Zero reconstruction error could be either local min or local max of density



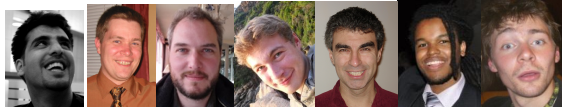
# Regularized Auto-Encoders Learn a Vector Field or a Markov Chain Transition Distribution

- (Bengio, Vincent & Courville, TPAMI 2013) review paper
- (Alain & Bengio ICLR 2013; Bengio et al, arxiv 2013)





# 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)

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

Training criterion:

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

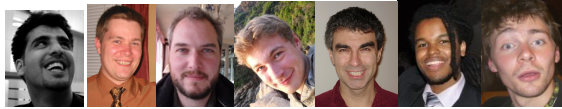
wants contraction in all directions

cannot afford contraction in manifold directions

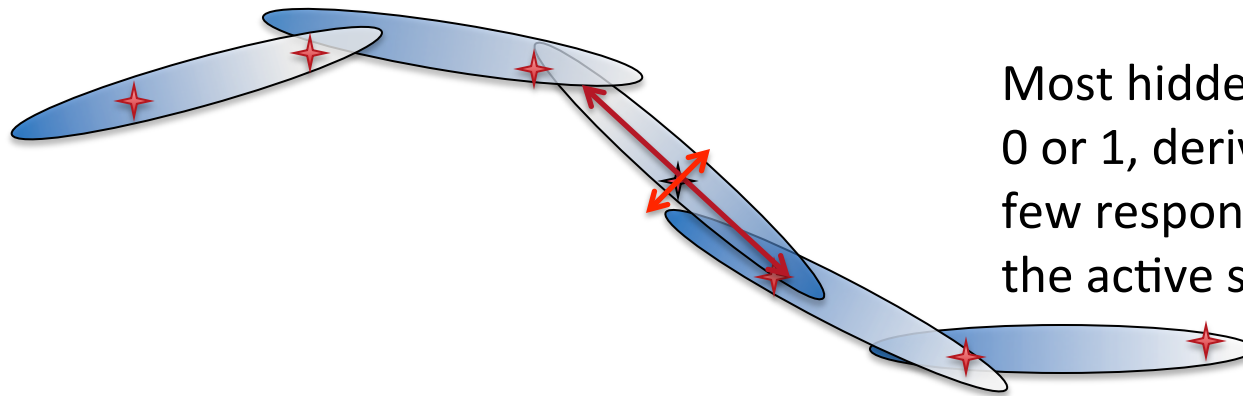
If  $h_j = \text{sigmoid}(b_j + W_j x)$

$$(\partial h_j(x) / \partial x_i)^2 = h_j^2 (1 - h_j)^2 W_{ji}^2$$

# 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** (near 0 or 1, derivative near 0):  
few responsive units represent  
the active subspace (local chart)

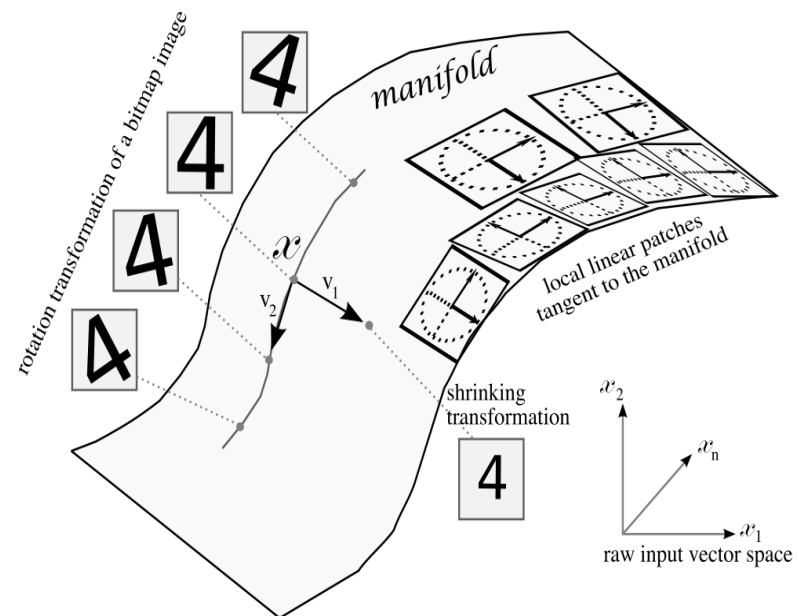
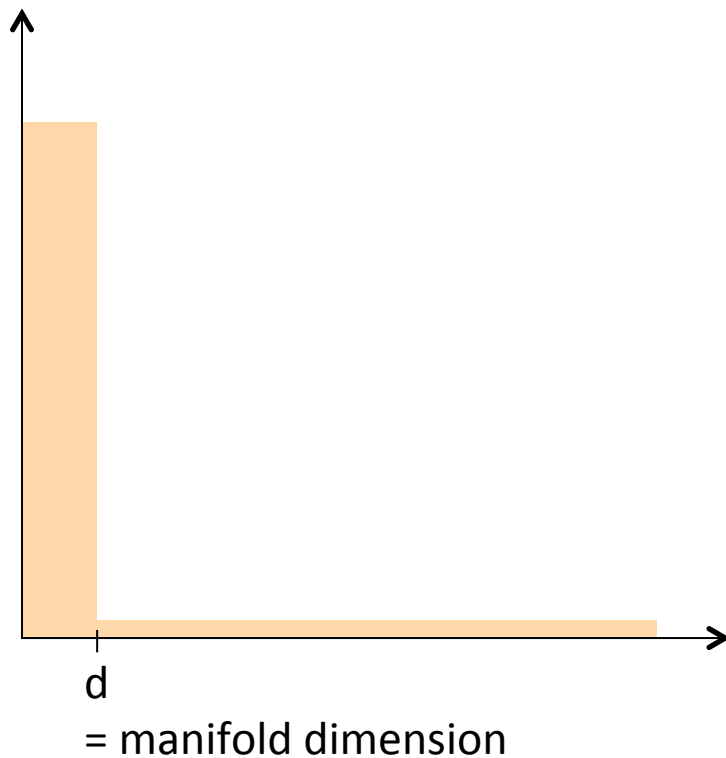
Each region/chart = subset of active hidden units

Neighboring region: one of the units becomes active/inactive

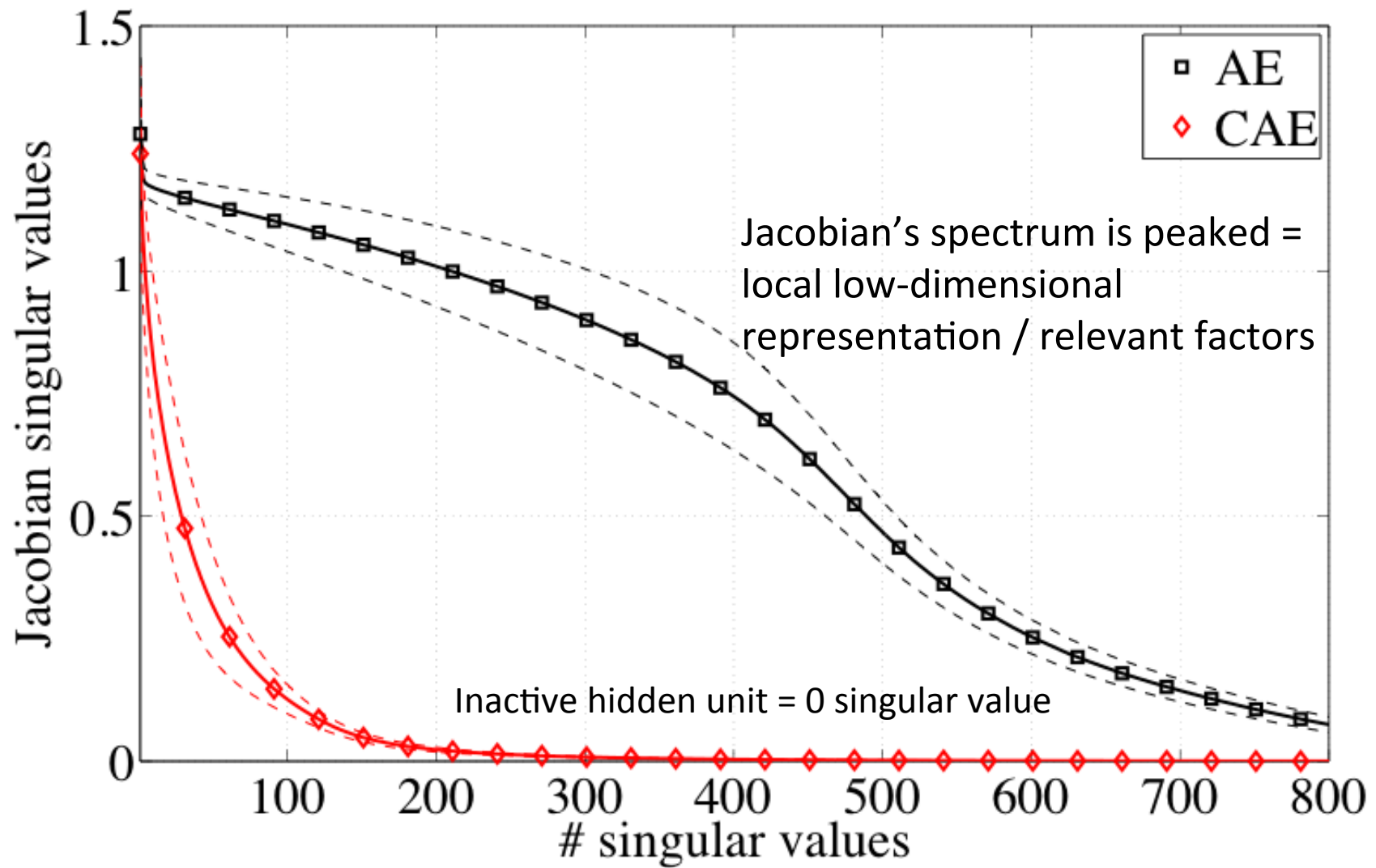
**SHARED SET OF FILTERS ACROSS REGIONS, EACH USING A SUBSET**

# Coordinate System & Eigenspectrum

- Ideal spectrum of  $dh/dx$  for manifolds



## CIFAR-10



Input Point

Tangents



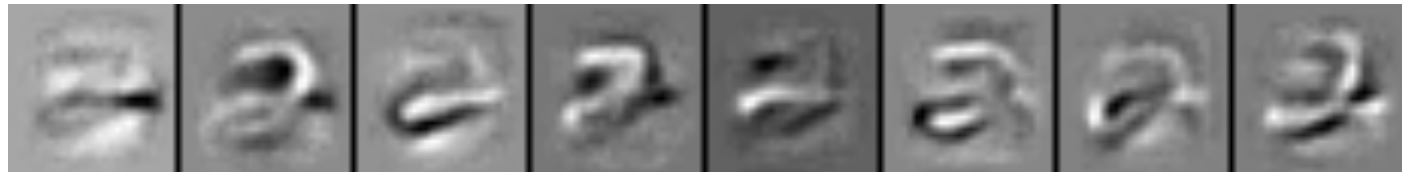
$$\text{Input Point} + 0.5 \times \text{Tangent} = \text{Result}$$

MNIST

Input Point



Tangents

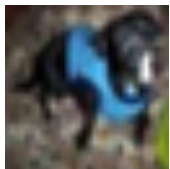


MNIST Tangents

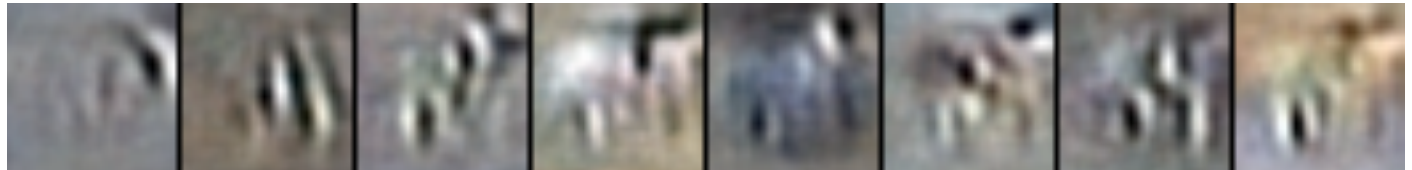
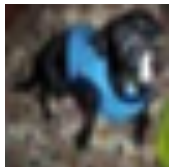
# Distributed vs Local (CIFAR-10 unsupervised)

Input Point

Tangents



Local PCA (no sharing across regions)



Contractive Auto-Encoder

# Denoising auto-encoders are also contractive!

- Taylor-expand Gaussian corruption noise in reconstruction error:

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

- Yields a contractive penalty in the **reconstruction function** (instead of encoder) proportional to amount of corruption noise



# Learned Tangent Prop: the Manifold Tangent Classifier

(Rifai et al NIPS 2011)

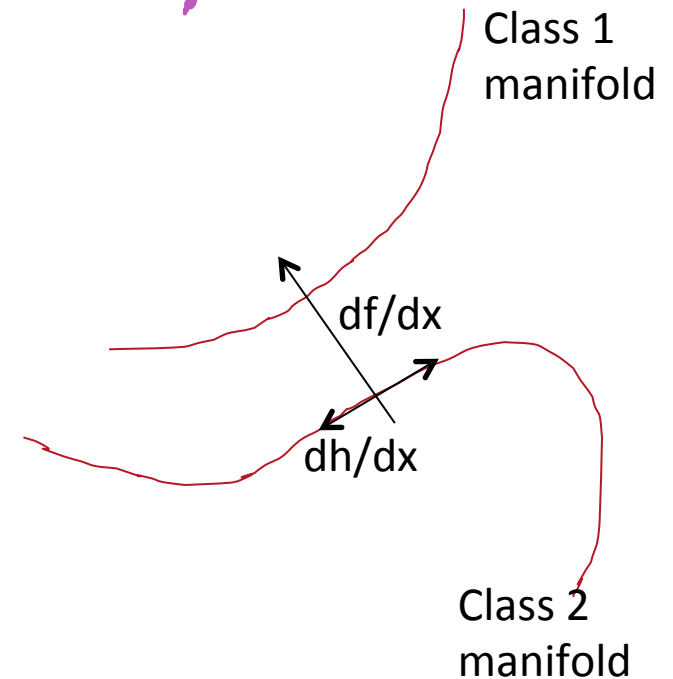
3 hypotheses:

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)

# Learned Tangent Prop: the Manifold Tangent Classifier

Makes classifier  $f(x)$  insensitive to variations on manifold at  $x$

Tangent plane characterized by  $dh(x)/dx$



(Rifai et al NIPS'2012)

# Learned Tangent Prop: the Manifold Tangent Classifier

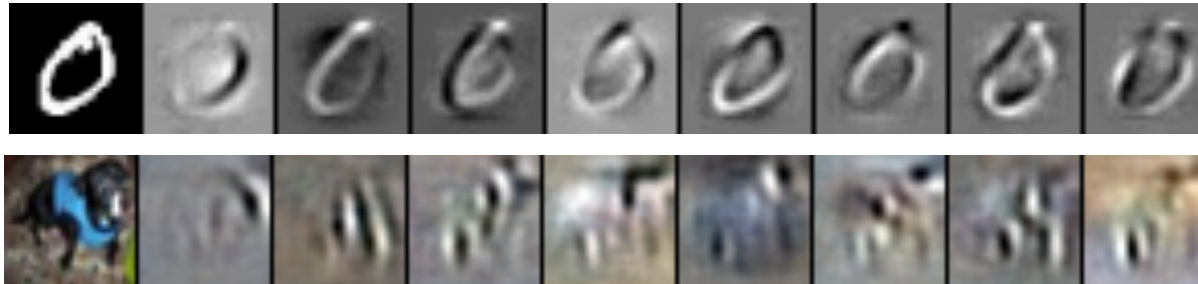
Algorithm:

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  $|| df/dx U(x) ||$

Makes  $f(x)$  insensitive to variations on manifold at  $x$ , tangent plane characterized by  $U(x)$ .

# Manifold Tangent Classifier Results

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



Trading  
&  
Markets

+gilt  
+yen  
+usda

-slow  
-term  
-debt

+matur  
+auction  
+treasur

-percent  
-sent  
-pressure

+bln  
+coupon  
+discount

-anti  
-predict  
-belgian

+interest  
+calcul  
+overnight

-sen  
-californ  
-introduc

- 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%	<b>0.81%</b>

- Semi-sup.

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

- Forest (500k examples)

SVM	Distributed SVM	MTC
4.11%	3.46%	<b>3.13%</b>

# Predictive Sparse Decomposition

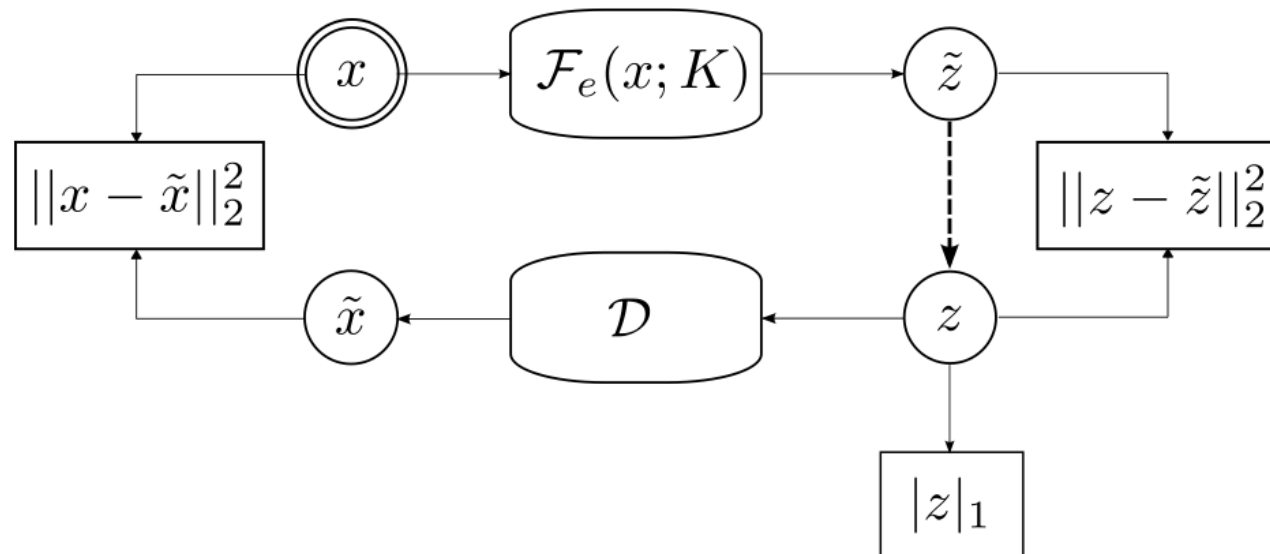
- Introduce an auxiliary variable in auto-encoder
- Approximate the inference of sparse coding by a parametric encoder:



## Predictive Sparse Decomposition

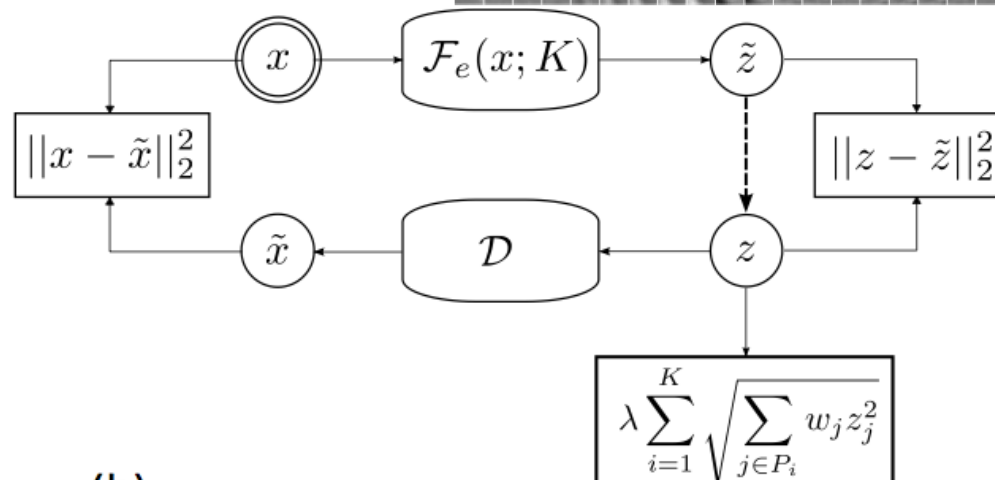
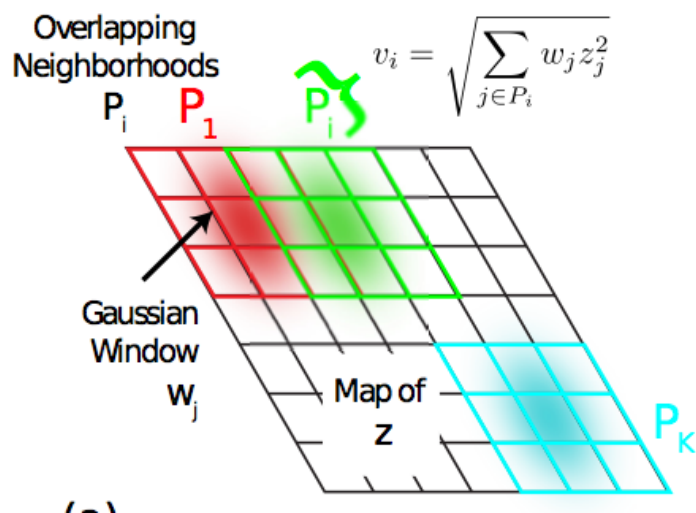
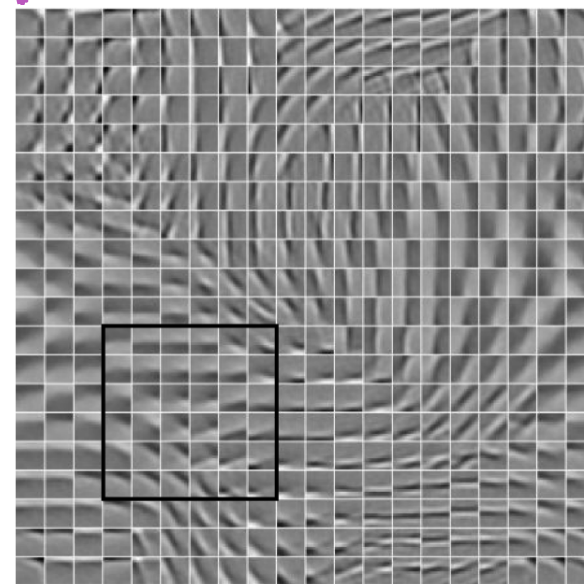
(Kavukcuoglu et al 2008)

- 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

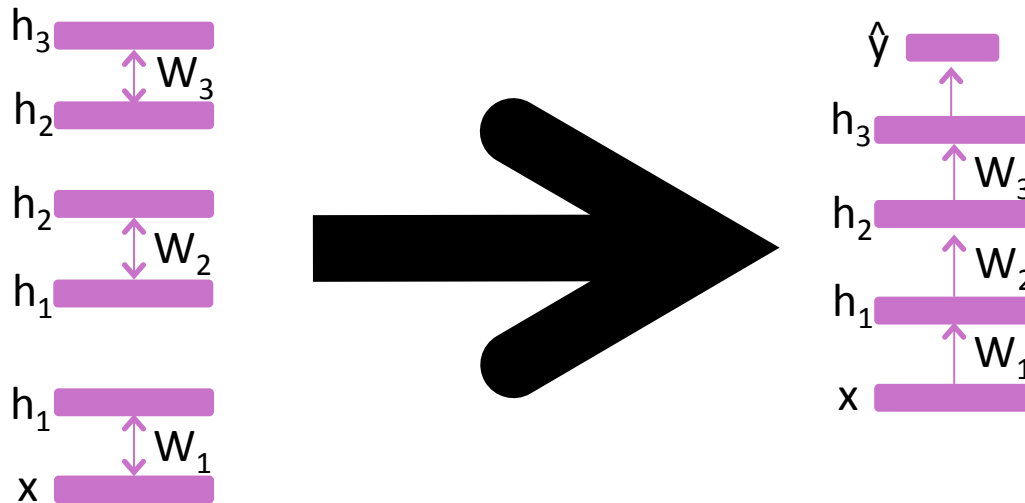
# Level-Local Learning is Important

- Initializing each layer of an unsupervised deep Boltzmann machine helps a lot
- Initializing each layer of a supervised neural network as an RBM, auto-encoder, denoising auto-encoder, etc can help a lot
- Helps most the layers further away from the target
- Not just an effect of the unsupervised prior
- Jointly training all the levels of a deep architecture is difficult because of the increased non-linearity / non-smoothness
- Initializing using a **level-local learning algorithm** is a useful trick
- Providing intermediate-level targets can help tremendously  
(Gulcehre & Bengio ICLR 2013)



# Stack of RBMs / AEs → Deep MLP

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

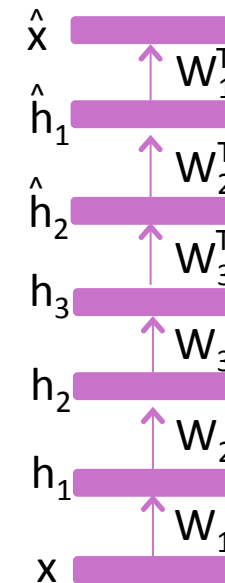
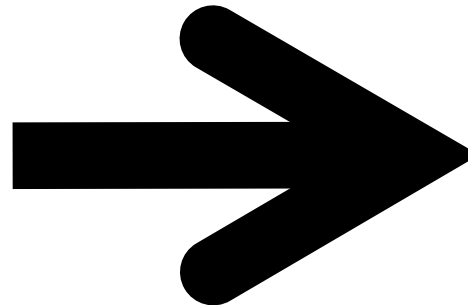
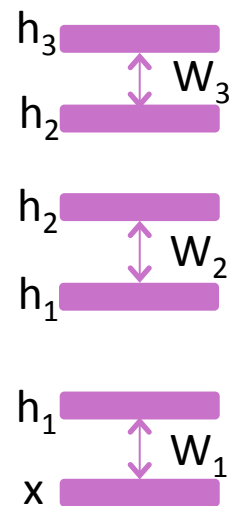


# Stack of RBMs / AEs → Deep Auto-Encoder

(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

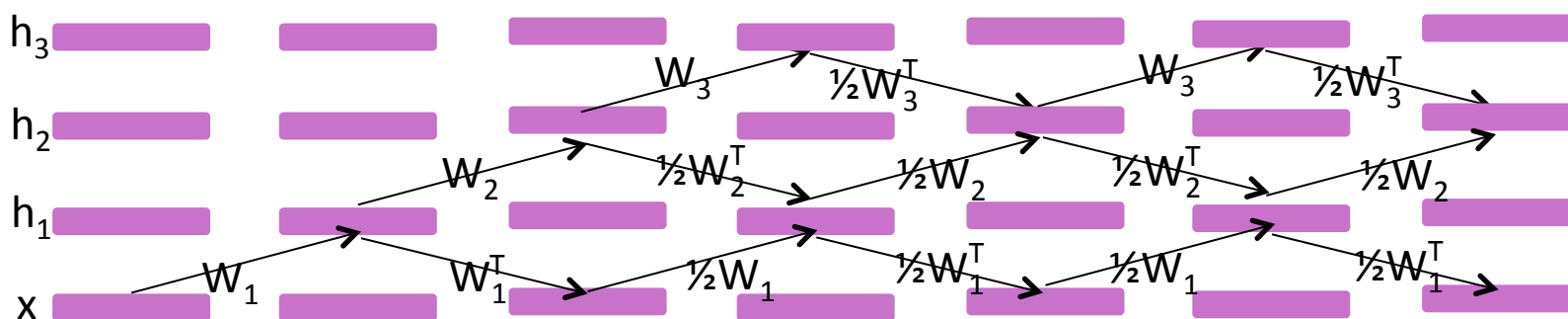
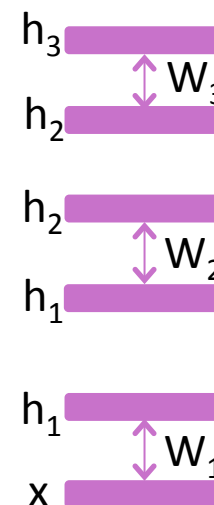
(Savard 2011)



(Bengio & Laufer, arxiv 2013)



- Each hidden layer receives input from below and above
- Deterministic (mean-field) recurrent computation (Savard 2011)
- Stochastic (injecting noise) recurrent computation: Deep Generative Stochastic Networks (GSNs)  
(Bengio & Laufer arxiv 2013)

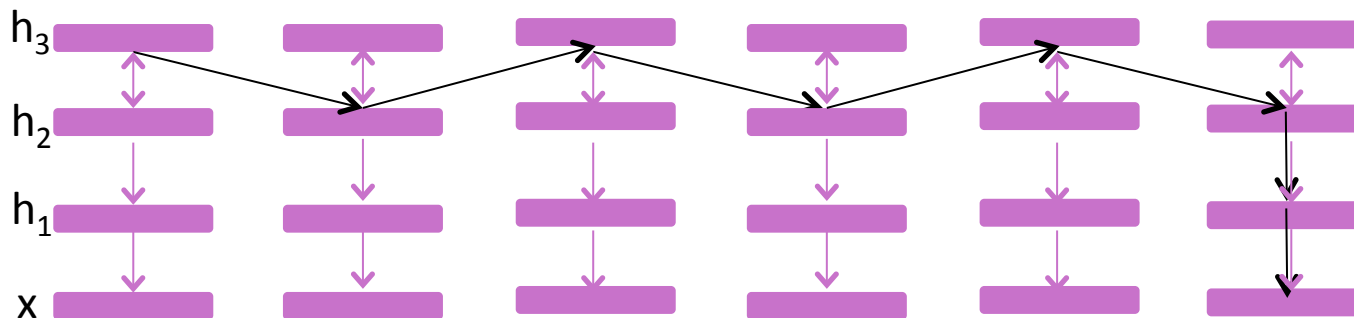


# Stack of RBMs → Deep Belief Net



(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



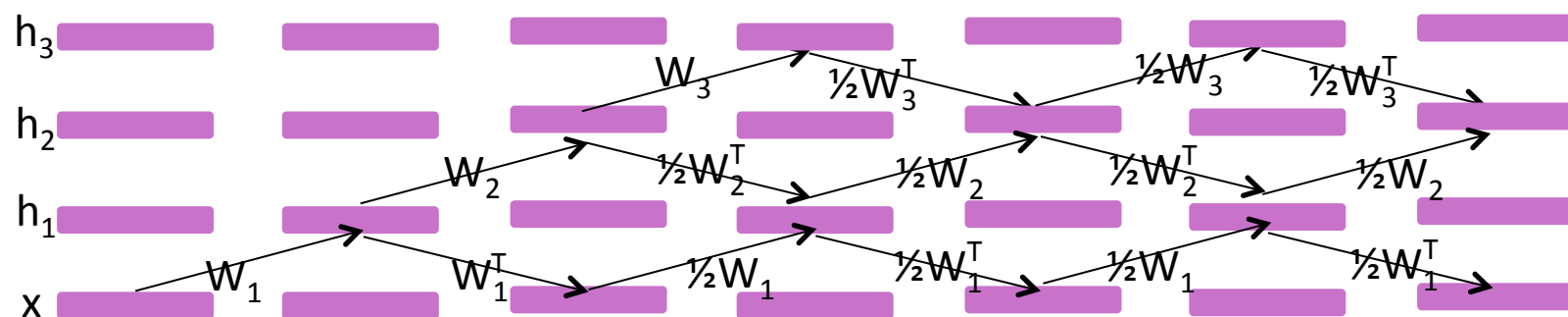


# Stack of RBMs

## → 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

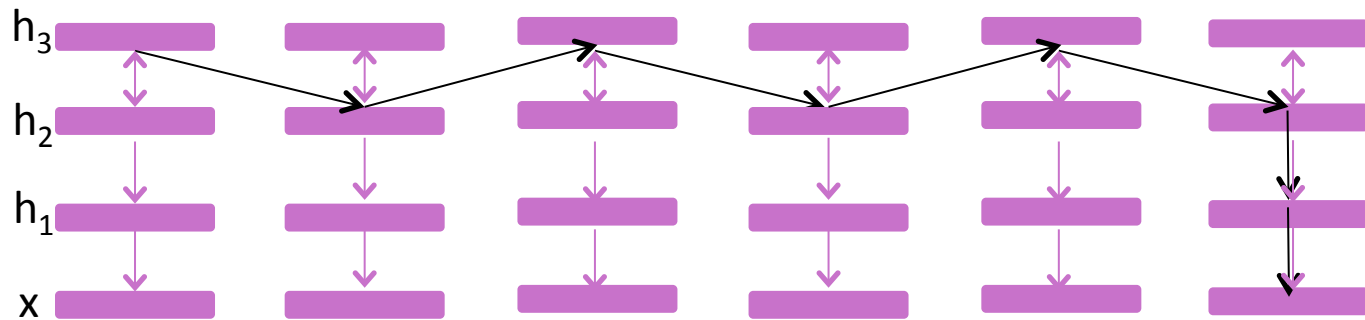


# Stack of Auto-Encoders → Deep Generative Auto-Encoder

(Rifai et al ICML 2012)



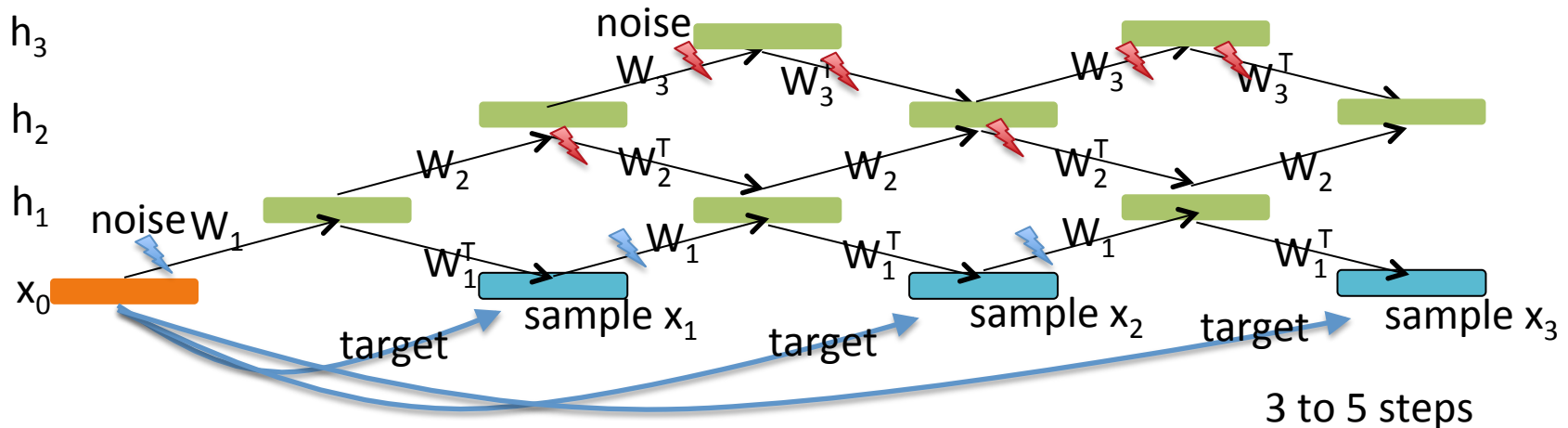
- MCMC on top-level auto-encoder
  - $h_{t+1} = \text{encode}(\text{decode}(h_t)) + \sigma \text{ noise}$   
where noise is  $\text{Normal}(0, d/dh \text{ encode}(\text{decode}(h_t)))$
- Then deterministically propagate down with decoders



# Generative Stochastic Networks (GSN)

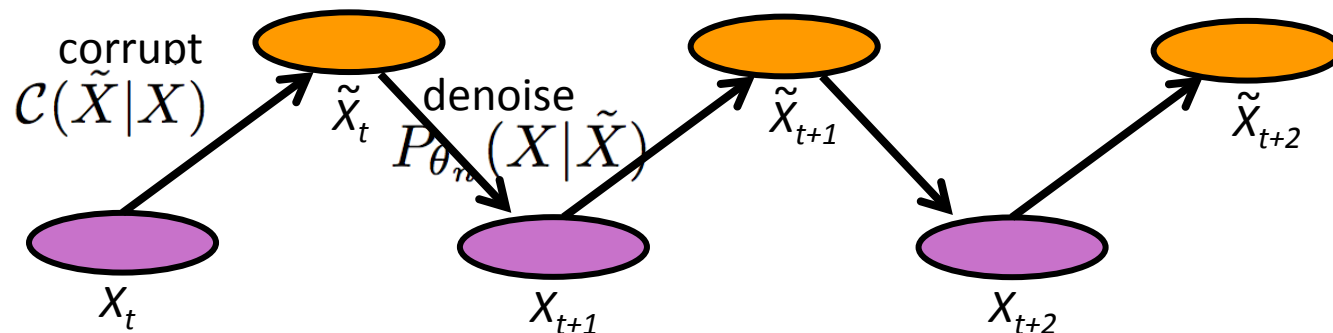
(Bengio, Yao, Alain & Vincent, arxiv 2013; Bengio & Laufer, arxiv 2013)

- **Recurrent parametrized stochastic computational graph that defines a transition operator for a Markov chain whose asymptotic distribution is implicitly estimated by the model**
- Noise injected in input and hidden layers
- Trained to max. reconstruction prob. of example at each step
- **Example** structure inspired from the DBM Gibbs chain:



# Denoising Auto-Encoder Markov Chain

- $\mathcal{P}(X)$ : true data-generating distribution
- $\mathcal{C}(\tilde{X}|X)$ : corruption process
- $P_{\theta_n}(X|\tilde{X})$ : denoising auto-encoder trained with  $n$  examples  $X, \tilde{X}$  from  $\mathcal{C}(\tilde{X}|X)\mathcal{P}(X)$ , probabilistically “inverts” corruption
- $T_n$ : Markov chain over  $X$  alternating  $\tilde{X} \sim \mathcal{C}(\tilde{X}|X)$ ,  $X \sim P_{\theta_n}(X|\tilde{X})$





# Previous Theoretical Results on Probabilistic Interpretation of Auto-Encoders

(Vincent 2011, Alain & Bengio 2013)

- Continuous  $X$
- Gaussian corruption
- Noise  $\sigma \rightarrow 0$
- Squared reconstruction error  $\|r(X+\text{noise})-X\|^2$

$(r(X)-X)/\sigma^2$  estimates the score  $d \log p(X) / dX$

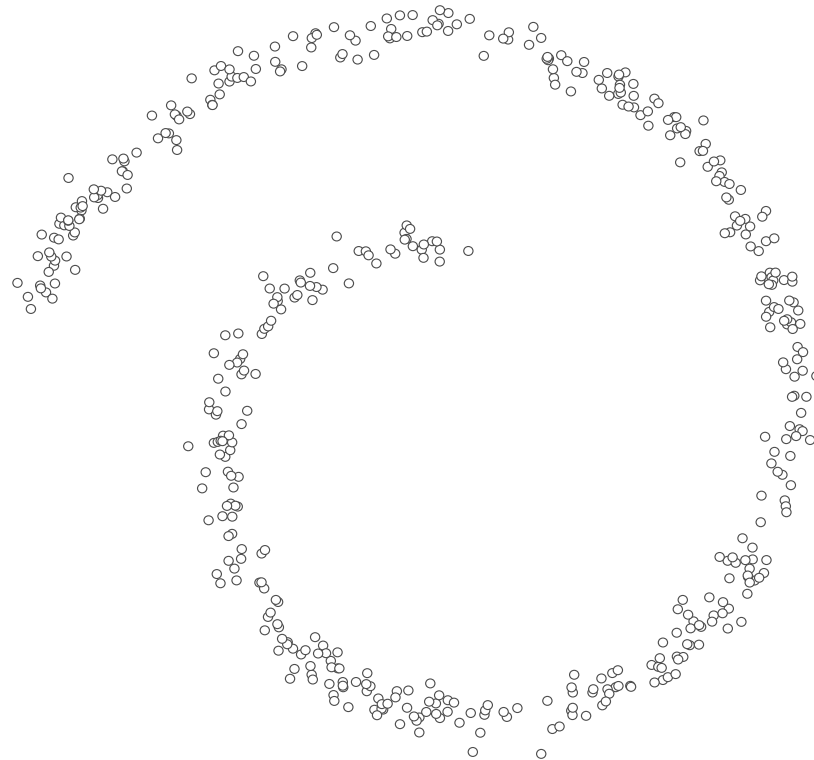
## New Theoretical Results (Bengio et al NIPS 2013)

- Denoising AE are consistent estimators of the data-generating distribution through their Markov chain, so long as they consistently estimate the conditional denoising distribution and the Markov chain converges.

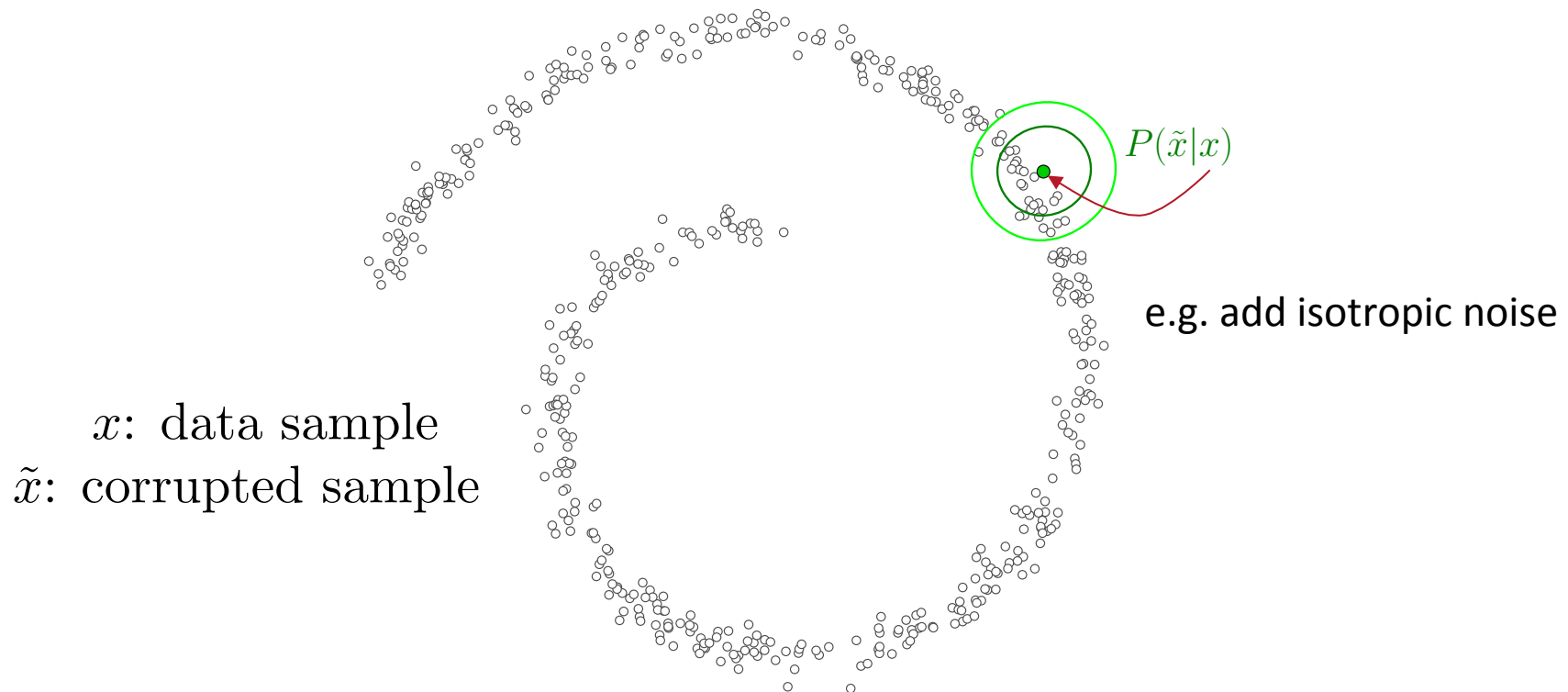
Making  $P_{\theta_n}(X|\tilde{X})$  match  $\mathcal{P}(X|\tilde{X})$  makes  $\pi_n(X)$  match  $\mathcal{P}(X)$

denoising distr.      truth      stationary distr.      truth

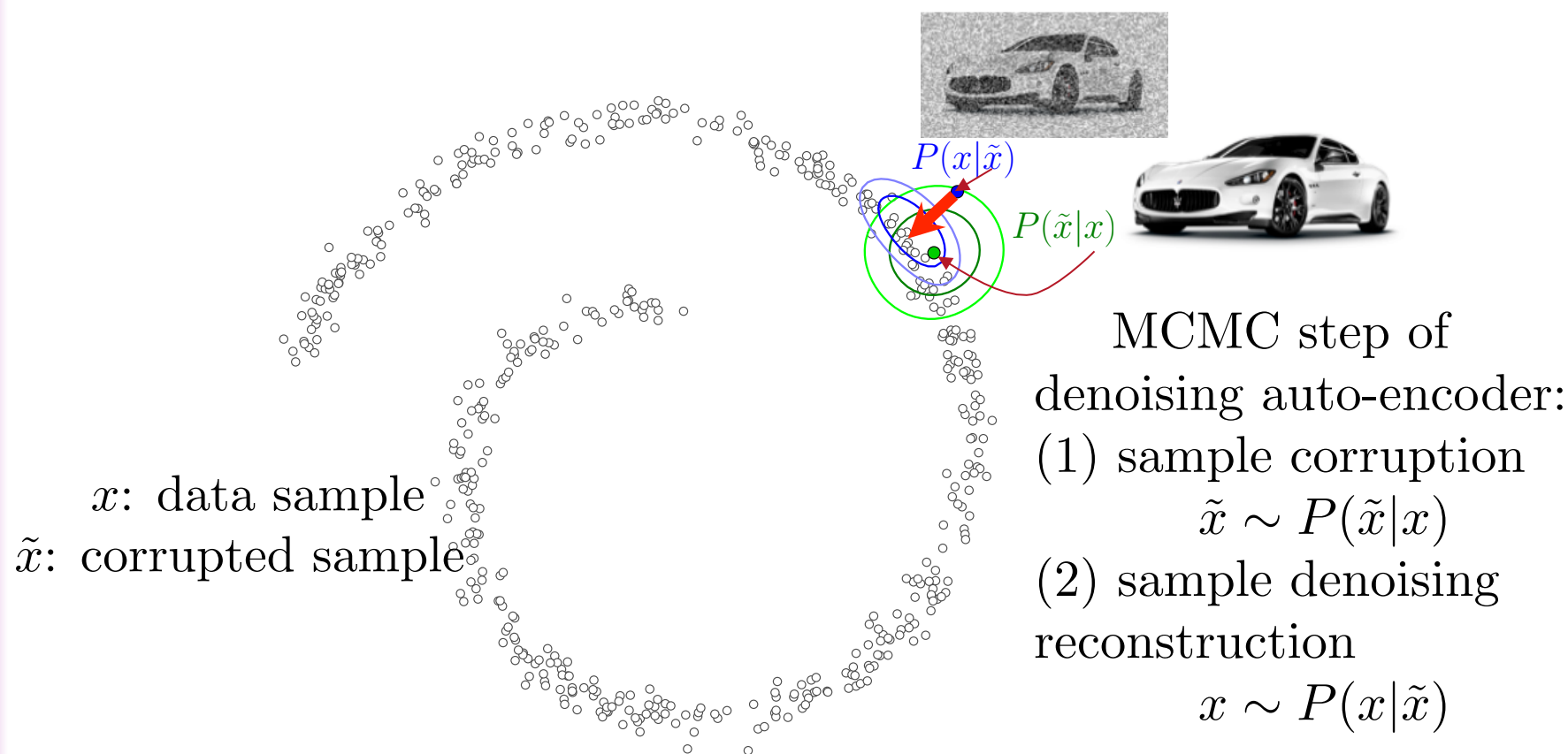
Learning with a simpler normalization constant, a nearly unimodal conditional distribution instead of a complicated multimodal one



Learning with a simpler normalization constant, a nearly unimodal conditional distribution instead of a complicated multimodal one

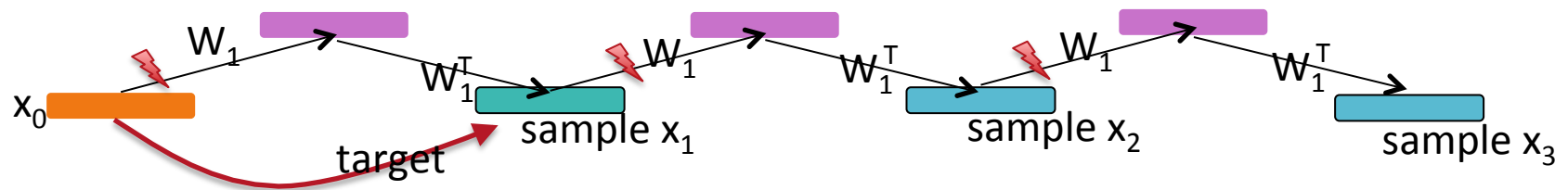


Learning with a simpler normalization constant, a nearly unimodal conditional distribution instead of a complicated multimodal one



## Shallow Model: Generalizing the Denoising Auto-Encoder Probabilistic Interpretation

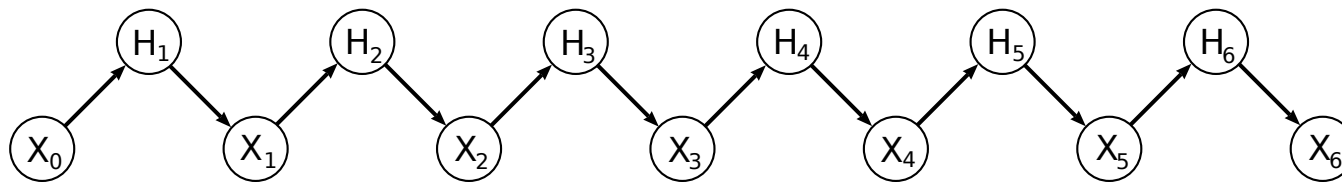
- Classical denoising auto-encoder architecture, single hidden layer with noise only injected in input
- Factored Bernoulli reconstruction prob. distr.
- $\tilde{X} = f_{\theta_1}(X, Z)$  = parameter-less, salt-and-pepper noise on top of  $X$



- *Generalizes (Alain & Bengio ICLR 2013): not just continuous r.v., any training criterion (as log-likelihood), not just Gaussian but any corruption (no need to be tiny to correctly estimate distribution).*

# Previous Work: Denoising Auto-Encoders as Generative Models

- (Bengio et al, NIPS'2013, Generalized denoising auto-encoders as generative models)
- State = data  $X$ , latent variable =  $H$  = corrupted data
- Parametrization: learn  $P(X | H)$ .
- Denoising: predict data  $X_0$  from corrupted version  $H_1$ , estimate  $P(X_0 | H_1)$ . Sample  $X_1$  accordingly, etc.

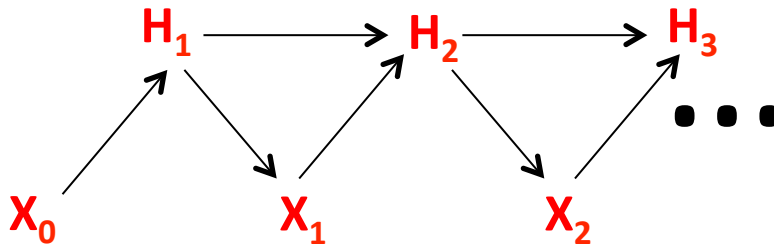


- **Theorem:** if the denoising estimator  $P(X | H)$  is consistent, then the corrupt+denoise Markov chain stationary distribution is a consistent estimator of  $P(X)$ .

# Generative Stochastic Networks

- Generalizes the denoising auto-encoder training scheme
  - Introduce latent variables in the Markov chain (over  $X, H$ )
  - Instead of a fixed corruption process, have a deterministic function with parameters  $\theta_1$  and a noise source  $Z$  as input

$$H_{t+1} = f_{\theta_1}(X_t, Z_t, H_t)$$



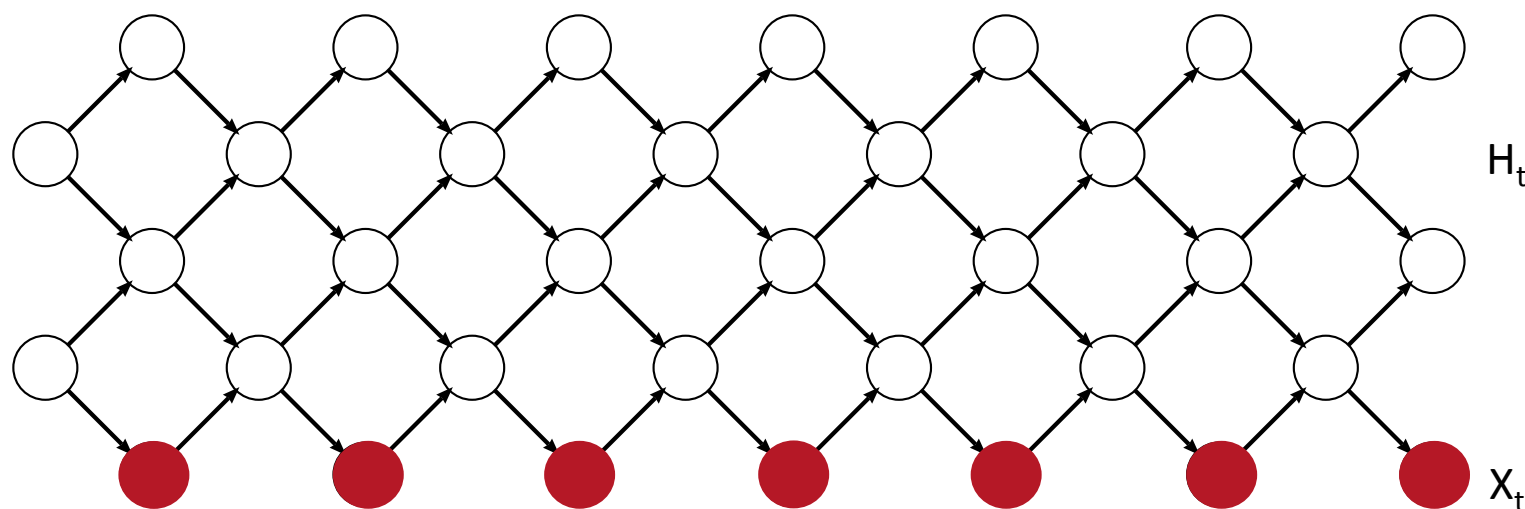
$$H_{t+1} \sim P_{\theta_1}(H|H_t, X_t)$$

$$X_{t+1} \sim P_{\theta_2}(X|H_{t+1})$$

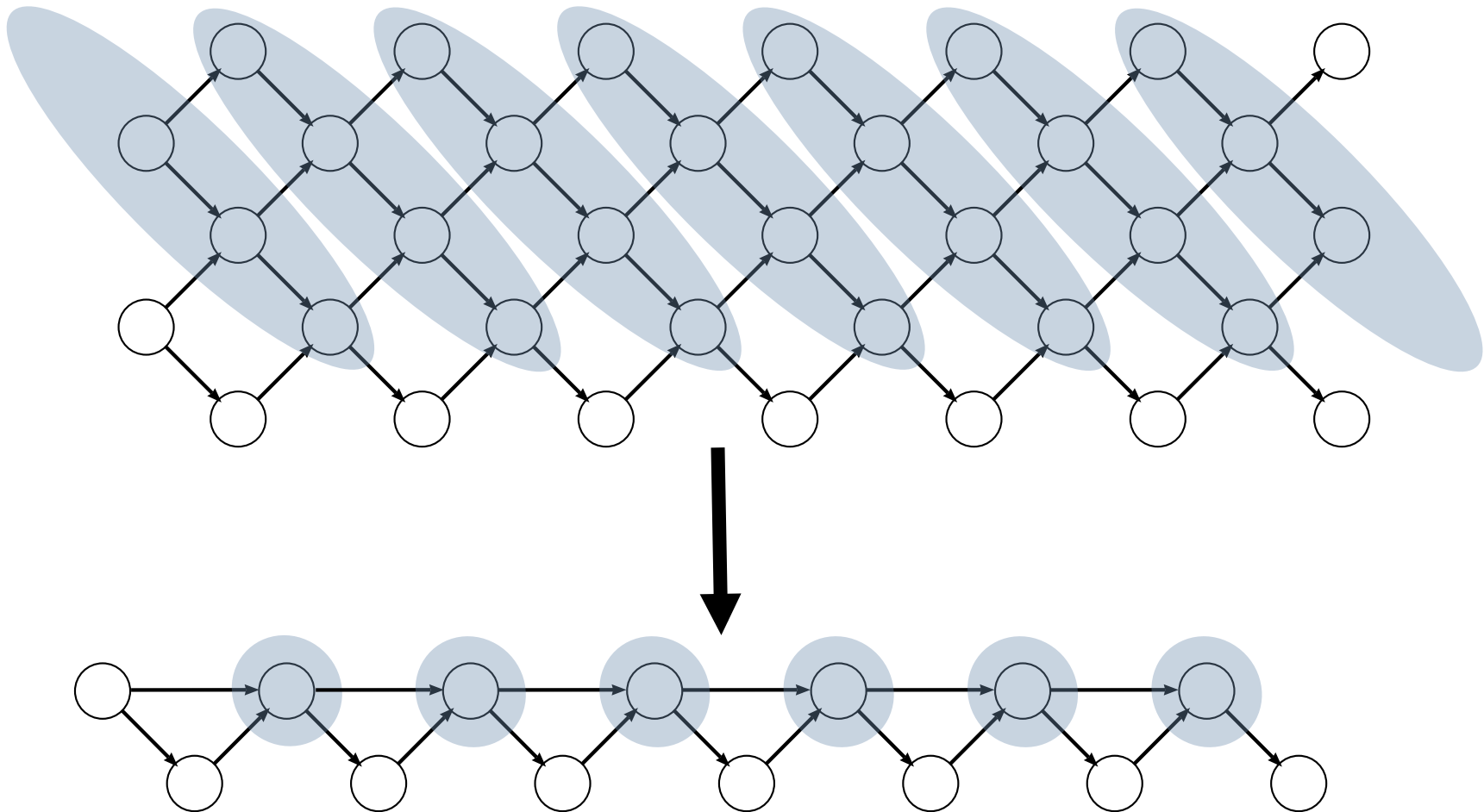


# Deep Generative Architecture

Allow multiple levels of latent variables with arbitrary (but differentiable) learned transformations in stochastic update function

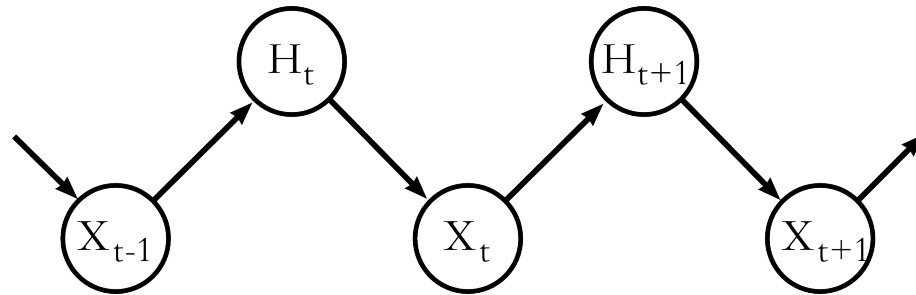


For theoretical analysis:  
Collapse the state

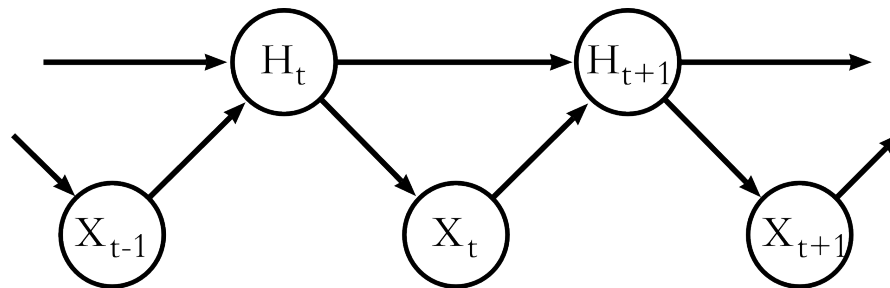


# Different Kinds of GSN Markov Chains

- Like Denoising Auto-Encoders:

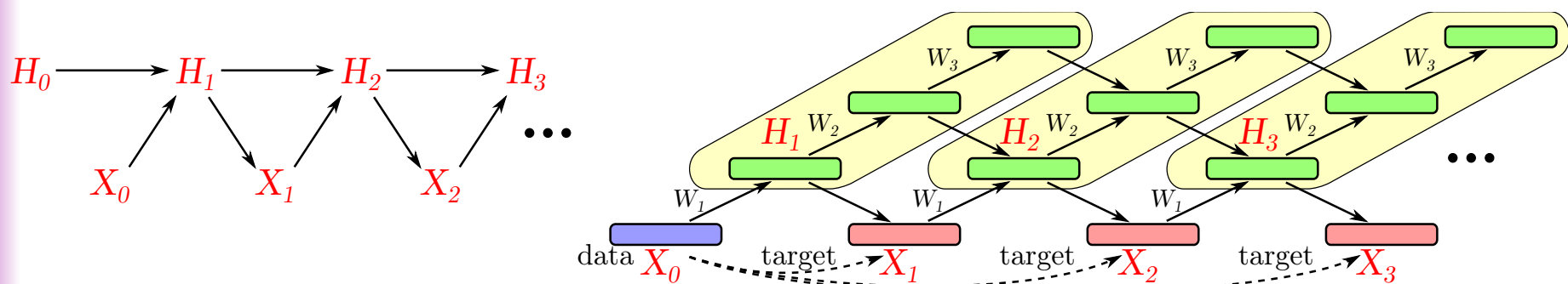


- With latent variables as necessary part of the state:



# Stochastic Recurrent Network Trainable by Backprop

- Using a state update scheme similar to block Gibbs sampling in the deep Boltzmann machine (Salakhdinov & Hinton 2009), but with continuous latent variables to be able to back-prop reconstruction error into it (**reparametrization trick**)



- Denoising-based training: maximize  $\log P(X_k=x_0 \mid H_k)$  and backprop into the net. Noise can be injected everywhere (must be injected for mixing to happen).

# Consistency Theorem

If we assume that the chain has a stationary distribution  $\pi_{H,X}$ , and that for every value of  $(x, h)$  we have that

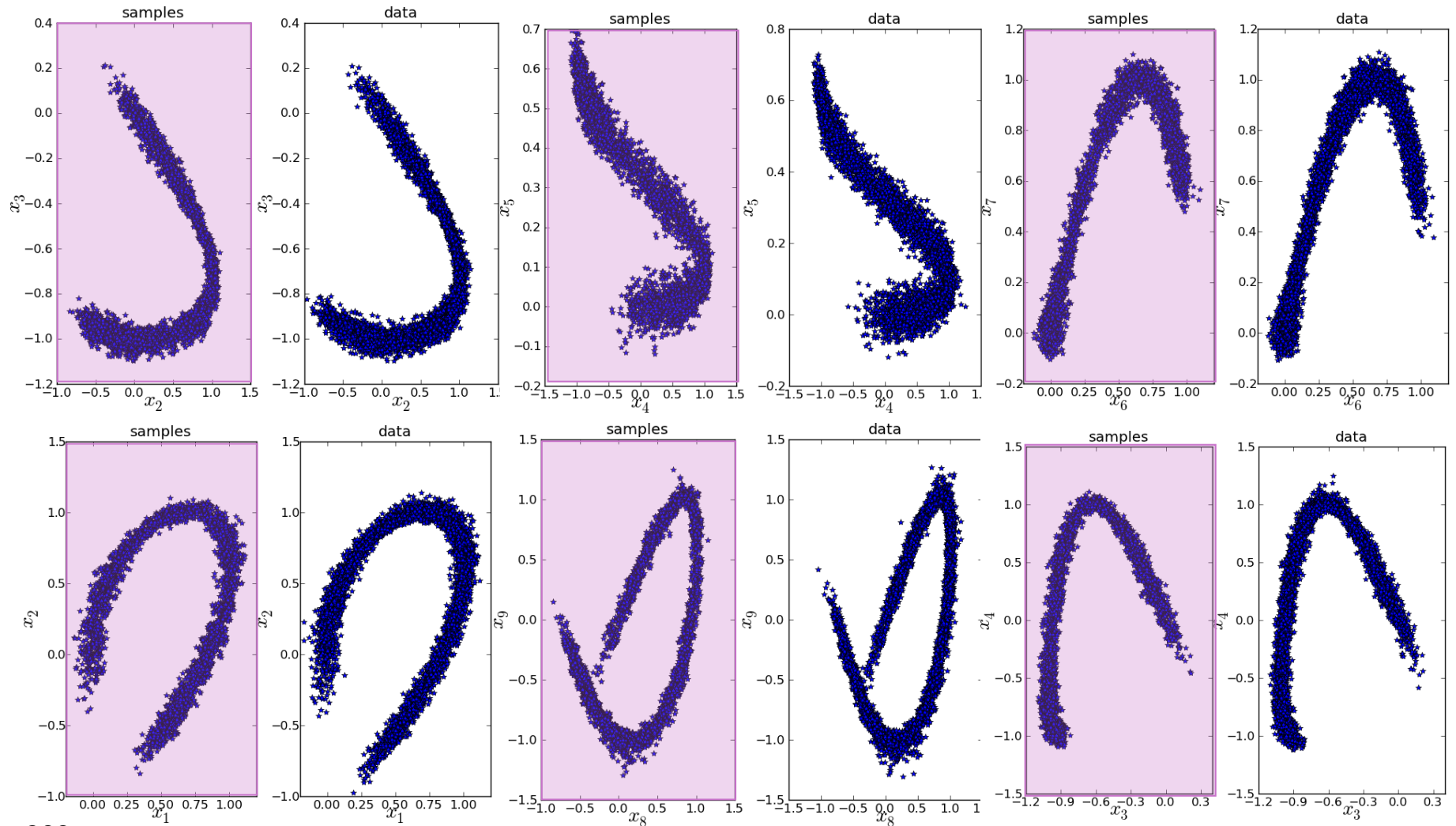
- all the  $P(X_t = x | H_t = h) = g(x, h)$  share the same density for  $t \geq 1$
  - all the  $P(H_{t+1} = h | H_t = h', X_t = x) = f(h, h', x)$  share the same density for  $t \geq 0$
  - $P(H_0 = h | X_0 = x) = P(H_1 = h | X_0 = x)$
  - $P(X_1 = x | H_1 = h) = P(X_0 = x | H_1 = h)$
- Time-invariant parametrization
- Initial state distr.=next state distr.
- Denoising is consistent

then for every value of  $(x, h)$  we get that

- $P(X_0 = x | H_0 = h) = g(x, h)$  holds, which is something that was assumed only for  $t \geq 1$
- $P(X_t = x, H_t = h) = P(X_0 = x, H_0 = h)$  for all  $t \geq 0$
- the stationary distribution  $\pi_{H,X}$  has a marginal distribution  $\pi_X$  such that  $\pi(x) = P(X_0 = x)$ .

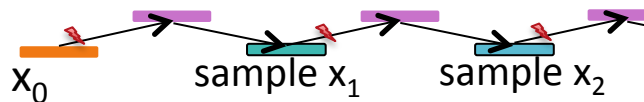
Those conclusions show that our Markov chain has the property that its samples in  $X$  are drawn from the same distribution as  $X_0$ .

# GSN Experiments: validating the theorem in a continuous non-parametric setting

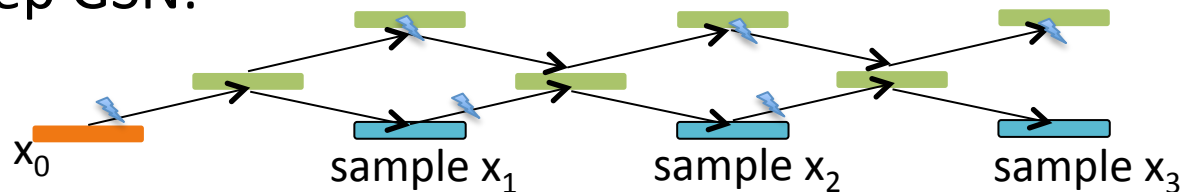


# Experiments: Shallow vs Deep

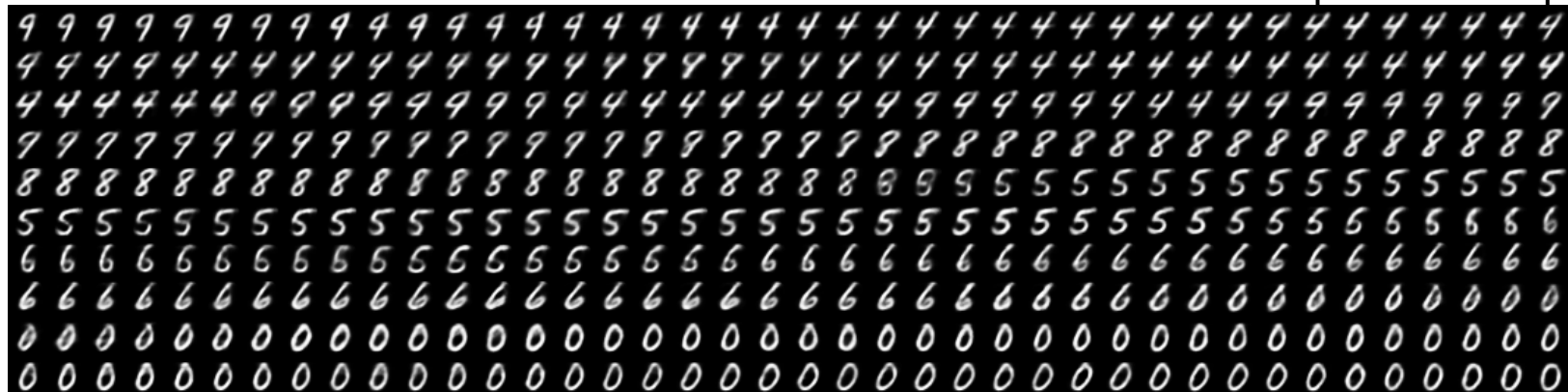
- Shallow (DAE), no recurrent path at higher levels, state=X only



- Deep GSN:



Better compromise  
between mixing and  
spurious samples





# (Un)conditional sampling




	GSN-2	DAE	DBN-2	CAE-1	CAE-2
LL	214	144	138	68	121
STD.ERR.	1.1	1.6	2.0	2.9	1.6



# Quantitative Evaluation of Samples

- Previous procedure for evaluating samples (Breuleux et al 2011, Rifai et al 2012, Bengio et al 2013):
  - Generate 10000 samples from model
  - Use them as training examples for Parzen density estimator
  - Evaluate its log-likelihood on MNIST test data



	GSN-2	DAE	RBM	DBM-3	DBN-2	MNIST
LOG-LIKELIHOOD	214	-152	-244	32	138	24
STANDARD ERROR	1.1	2.2	54	1.9	2.0	1.6

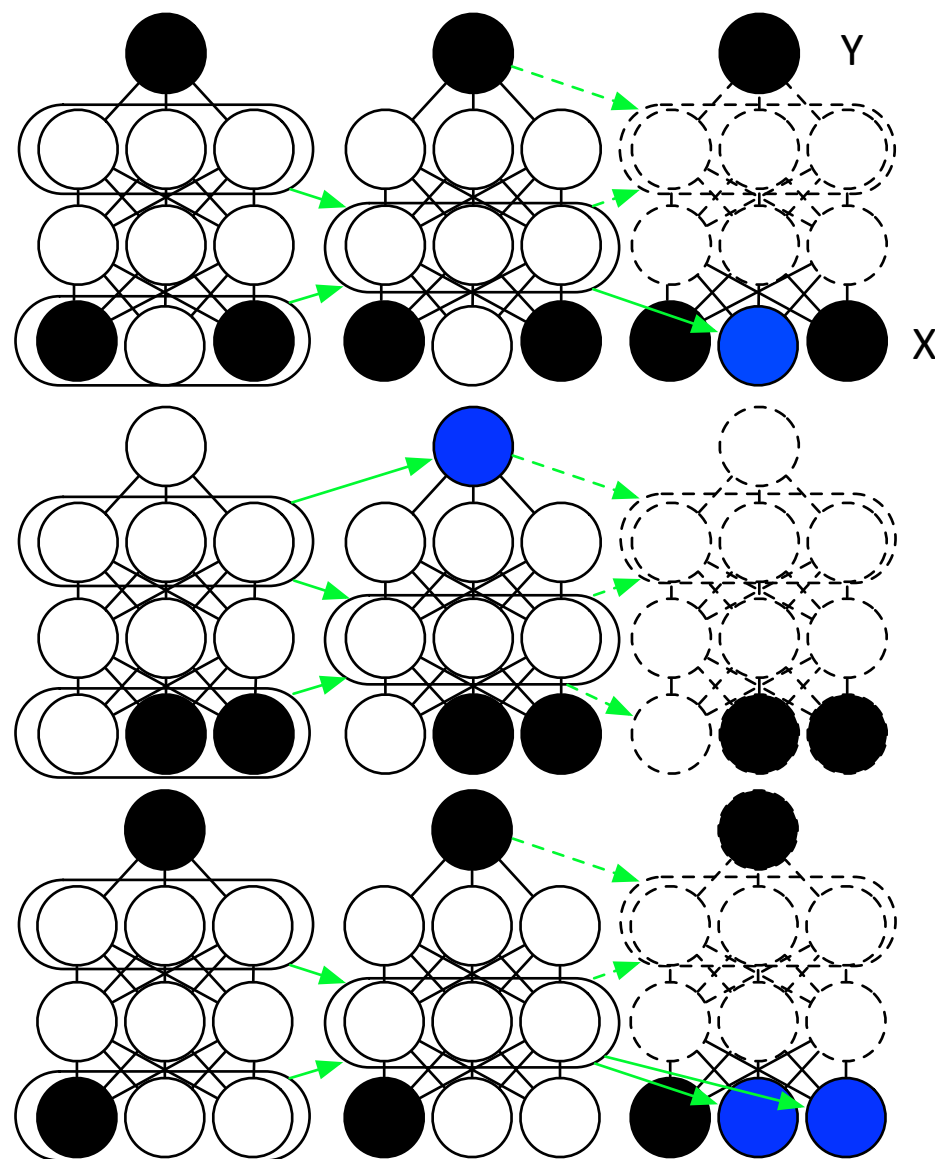
# A Proper Generative Model for Dependency Networks, MP-DBMs, and efficient deep NADE sampling

- Dependency nets (Heckerman et al 2000) estimate  $P_{\theta_i}(X_i | X_{-i})$  not guaranteed to be conditionals of a unique joint
- GSN defines a unique joint distribution = stationary distr. of chain (which averages out over resampling orders)
- Generalizes to composite likelihood:  $P(\text{subset}(X) | X \setminus \text{subset}(X))$  Justifies efficient sampling scheme for MP-DBMs and deep NADE.

4	5	9	9	7	5		7	9	8
1	4	-	8	2	9	4	0	0	5
0	7	9	0	8	5	/	5	1	9
9	9		4	2	8	9	8	9	6
2	9	9	9	9	4	9	5	4	9
9	9	6	9	8				6	1
5		/	5	8			1		1
/	6			8	9	6	6	/	1
5	9	9	6	9	8	5	4	8	2
8	6	9	2	4	8	9	1		

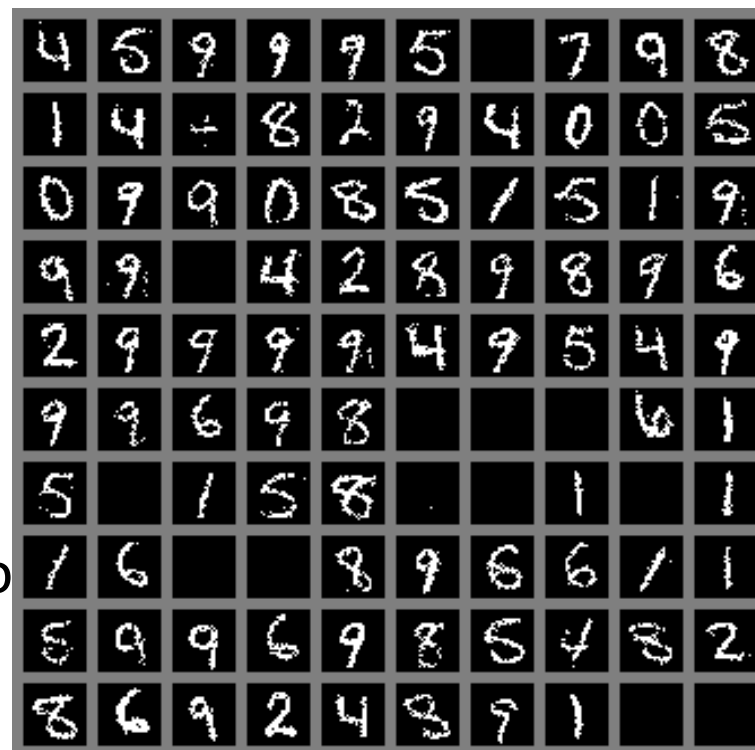
# MP-DBMs: GSNs for Classification (NIPS'2013)

- DBM + predict any subset of visibles given the others (generalized pseudo-likelihood), via mean-field recursion
- GSN MCMC sampling procedure: sample random subset of visibles, given the others; iterate.
- Classification: predict  $Y|X$



## MP-DBM Results

- Single model of (X,Y) vs multiple stages of training DBM + fine-tuning
- SOTA on permutation-invariant MNIST (at time of submission):
  - 0.88% error
- Salakhutdinov & Hinton's DBM: 0.95%
- NORB: 10.6% (vs 10.8% with S&H's DBM)
- DBM (Gibbs) samples of trained MP-DBM are ugly, while GSN sampling works because it better corresponds to the training criterion:



# Reparametrizing latent variables

- Insight from (Bengio et al 2013, arxiv 1306.1091 & 1308.3432) papers on GSNs and stochastic neurons:
  - Sampling from continuous latent variables (given some ancestors) can be rewritten as a deterministic function of other variables and of independent noise sources:  $h = f(x; \eta)$
  - This enables rewriting the gradient log-likelihood as back-prop, averaged over samples of the noise sources

$$P(y|x) = \int_h P(y|h, x)P(h|x)dh = \int_{\eta} P(y|f(x; \eta), x)P(\eta)d\eta$$
$$\frac{\partial P(y|x)}{\partial \theta} = \int_{\eta} \frac{\partial P(y|f(x; \eta), x)}{\partial \theta} P(\eta)d\eta$$

- A deeper formal analysis of this approach:
  - Kingma & Welling 2014, arxiv 1402.0480; see also Wierstra et al 2014, arxiv 1401.4082.

# Learned Approximate Inference

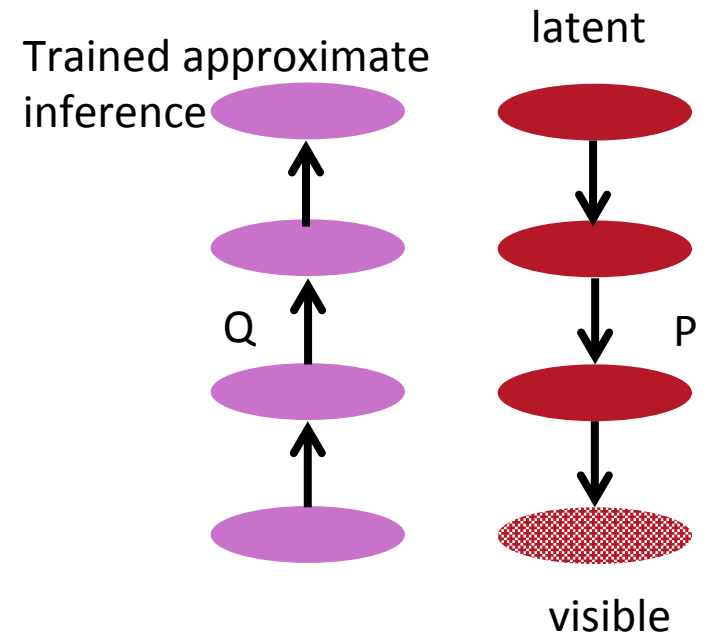
1. *Construct a computational graph corresponding to inference*
  - Loopy belief prop. (Ross et al CVPR 2011, Stoyanov et al 2011)
  - Variational mean-field (Goodfellow et al, ICLR 2013)
  - MAP (Kavukcuoglu et al 2008, Gregor & LeCun ICML 2010)
  - Proposal distribution / recognition net in Wake-Sleep algorithm (Hinton et al 1995)
2. *Optimize parameters wrt criterion of interest, possibly decoupling from the generative model's parameters*

Learning can compensate for the inadequacy of approximate inference, taking advantage of specifics of the data distribution

# Auto-Encoding Variational Bayes

(Kingma & Welling, ICLR 2014; DeepMind 2014)

- Revisiting the wake-sleep algorithm
- Generative model = deep net with injected noise: **decoder**
- Learned approximate inference = deep net with injected noise: **encoder**
- Latent variables are continuous, allowing to back-prop through (trick from GSN paper, [Bengio et al ICML 2014](#)) and train encoder & decoder jointly



# Applications

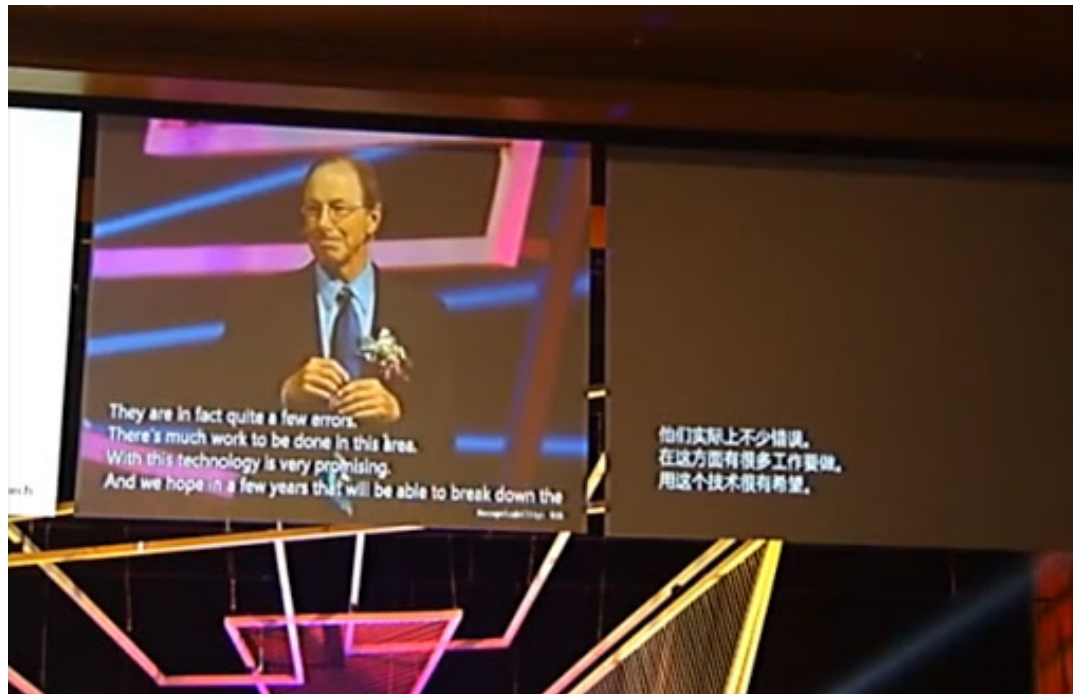


# AI Tasks

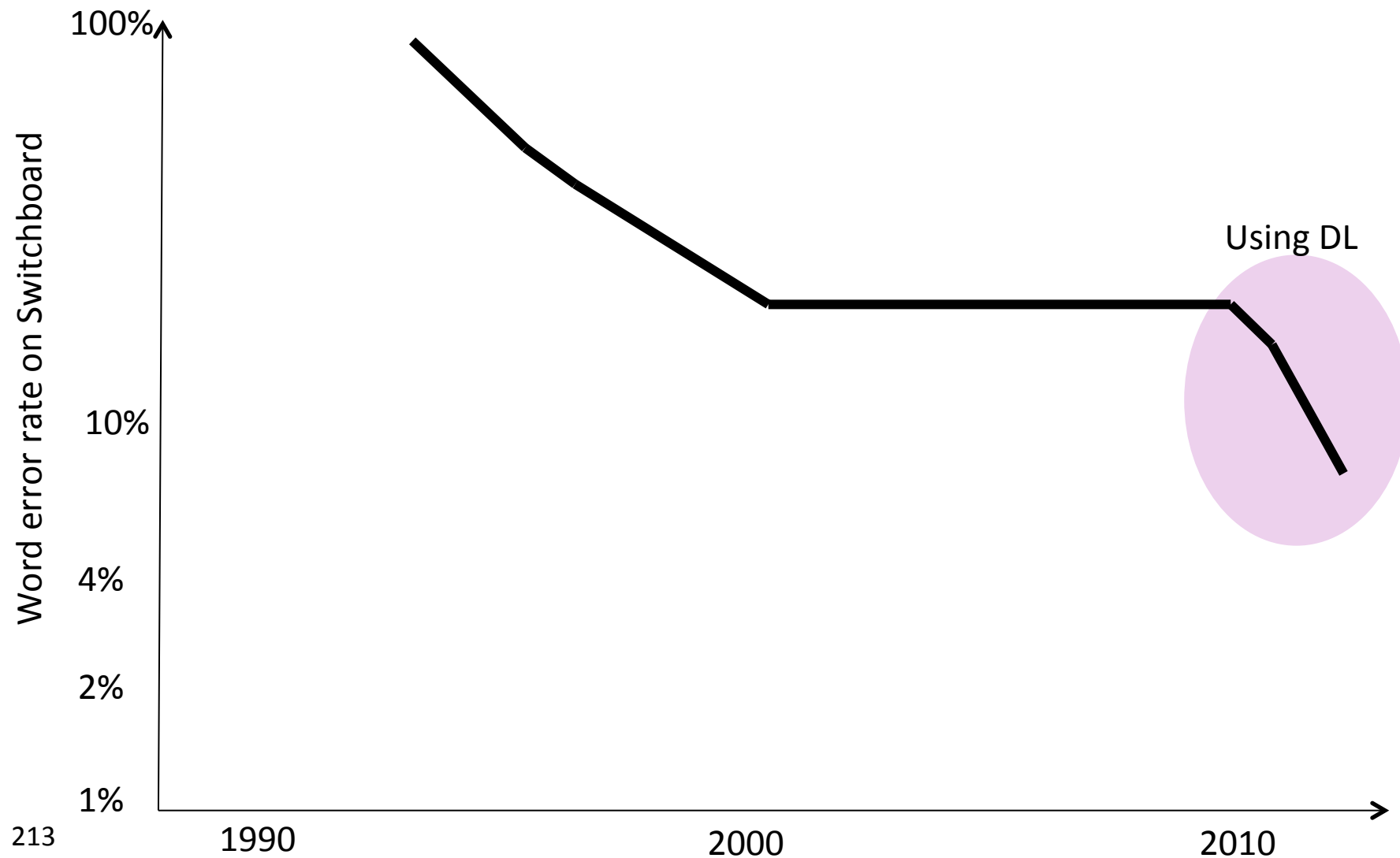
- Perception
  - Vision
  - Speech
  - Multiple modalities
- Natural language understanding
- Reinforcement learning & control
- COMPLEX HIGHLY-STRUCTURED DISTRIBUTION
- LOTS OF DATA (maybe mostly unlabeled)

## 2012: Industrial-scale success in speech recognition

- Google uses DL in their android speech recognizer (both server-side and on some phones with enough memory)
- Microsoft uses DL in their speech recognizer
- Error reductions on the order of 30%, a major progress



# The dramatic impact of Deep Learning on Speech Recognition (according to Microsoft)



# Deep Networks for Speech Recognition: results from Google, IBM, Microsoft

task	Hours of training data	Deep net+HMM	GMM+HMM same data	GMM+HMM more data
Switchboard	309	16.1	23.6	17.1 (2k hours)
English Broadcast news	50	17.5	18.8	
Bing voice search	24	30.4	36.2	
Google voice input	5870	12.3		16.0 (lots more)
Youtube	1400	47.6	52.3	

# Some Applications of DL

- **Language Modeling** (Speech Recognition, Machine Translation)
- **Acoustic Modeling** (**speech recognition**, music modeling)
- **NLP syntactic/semantic tagging** (Part-Of-Speech, chunking, Named Entity Recognition, Semantic Role Labeling, Parsing)
- **NLP applications**: sentiment analysis, paraphrasing, question-answering, Word-Sense Disambiguation
- **Object recognition in images**: photo search and image search: handwriting recognition, document analysis, handwriting synthesis, **superhuman traffic sign classification**, street view house numbers, **emotion detection from facial images**, roads from satellites.
- **Personalization**/recommendation/fraud/ads
- **Molecular properties**: QSAR, quantum calculations

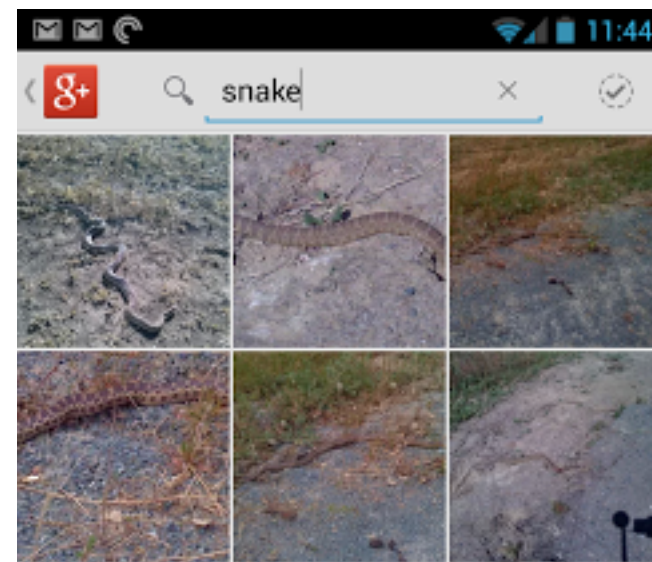


# Industrial-scale success in object recognition

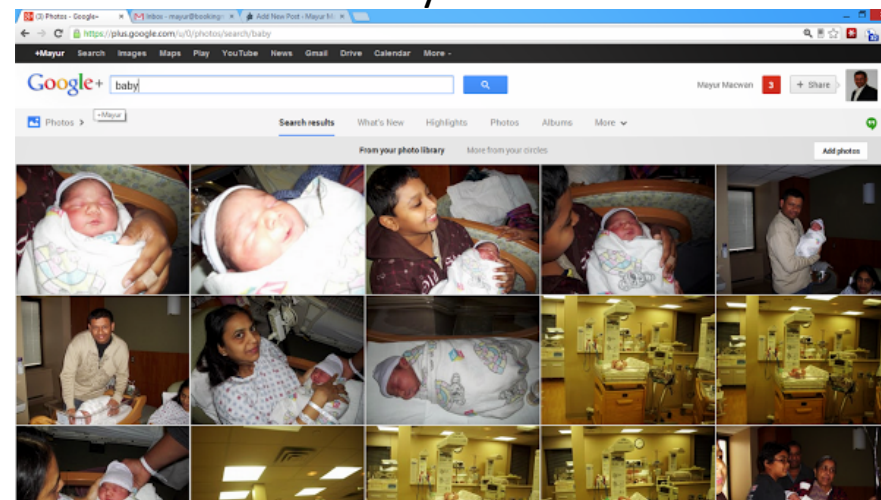
- Krizhevsky, Sutskever & Hinton NIPS 2012

	1 <sup>st</sup> choice	Top-5
2 <sup>nd</sup> best		27% err
Previous SOTA	45% err	26% err
Krizhevsky et al	37% err	15% err

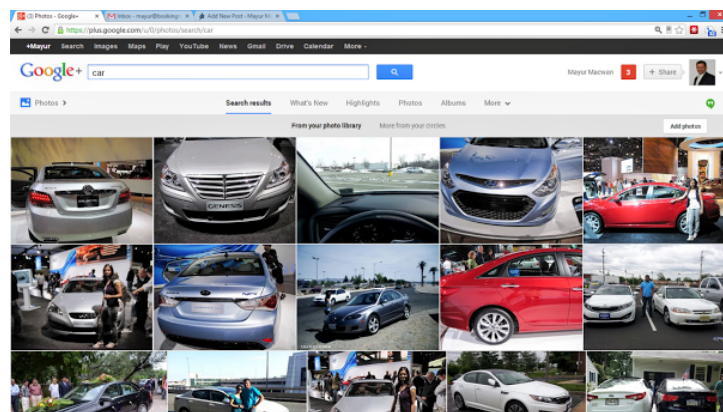
- Google incorporates DL in Google+ photo search, “A step across the semantic gap” (Google Research blog, June 12, 2013)**
- Baidu now offers similar services



baby



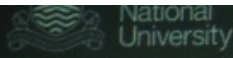
car





# Montreal Deep Nets Win Emotion Recognition in the Wild Challenge

Predict emotional expression from video (using images + audio)



**Results!**

Team Name	Classification accuracy	
Audio baseline	22.4 %	
Video baseline	22.7 %	
Fusion	27.5 %	
Nottingham	24.7 %	
Oulu	21.5 %	
KIT	29.8 %	
UCSD	37.1 %	2nd
ICT@CAS	35.9 %	3rd
York	27.6 %	
LNMIIT	20.5 %	
Montreal	41.0 %	1st
Ulm	27.2 %	

Dec. 9, 2013

# More Successful Applications

- Microsoft uses DL for speech rec. service (audio video indexing), based on Hinton/Toronto's DBNs (Mohamed et al 2012)
- Google uses DL in its Google Goggles service, using Ng/Stanford DL systems, and in its Google+ photo search service, using deep convolutional nets
- NYT talks about these: [http://www.nytimes.com/2012/06/26/technology/in-a-big-network-of-computers-evidence-of-machine-learning.html?\\_r=1](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



# Already Many NLP Applications of DL

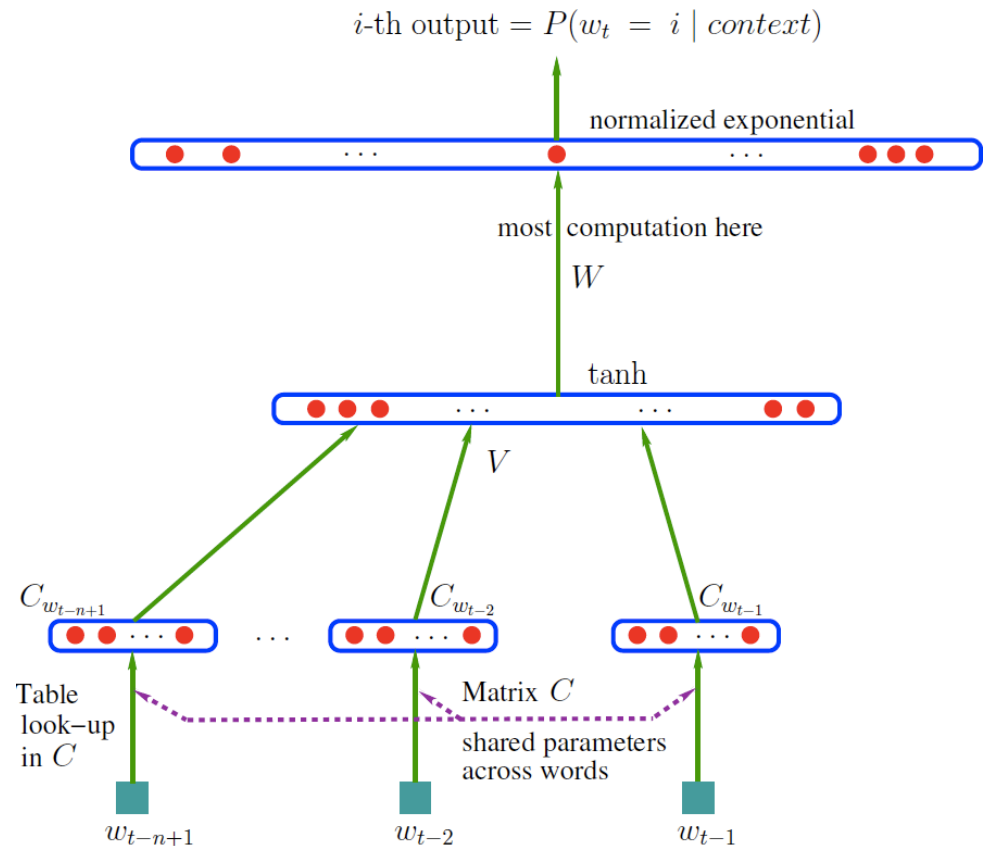
- Language Modeling (Speech Recognition, Machine Translation)
- Acoustic Modeling
- Part-Of-Speech Tagging
- Chunking
- Named Entity Recognition
- Semantic Role Labeling
- Parsing
- Sentiment Analysis
- Paraphrasing
- Question-Answering
- Word-Sense Disambiguation

# Neural Language Model

- *Bengio et al NIPS'2000 and JMLR 2003 "A Neural Probabilistic Language Model"*



- Each word represented by a distributed continuous-valued code vector = embedding
- Generalizes to sequences of words that are **semantically similar** to training sequences

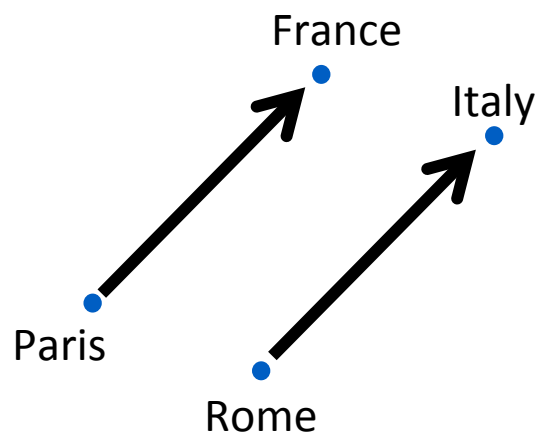


# Neural word embeddings - visualization

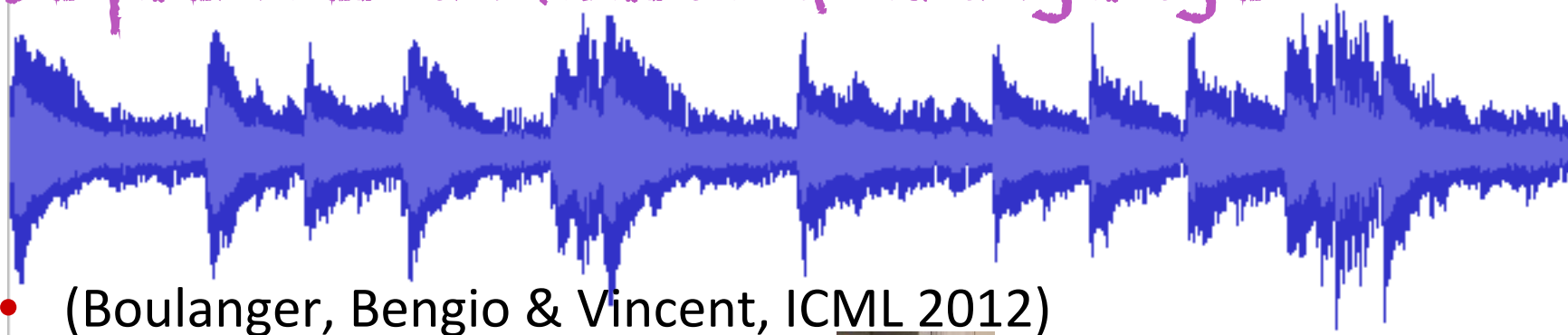


# Analogical Representations for Free (Mikolov et al, ICLR 2013)

- Semantic relations appear as linear relationships in the space of learned representations
- King – Queen  $\approx$  Man – Woman
- Paris – France + Italy  $\approx$  Rome



# Deep / Recurrent Nets for Modeling Sequences in Music & Language



- (Boulanger, Bengio & Vincent, ICML 2012)

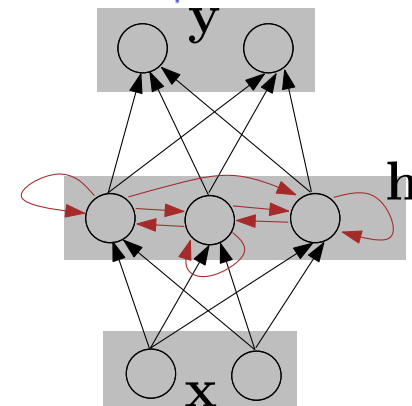
- Recurrent nets + RBMs

**SOTA** • Acoustics  $\rightarrow$  musical score



- (Bengio, Boulanger & Pascanu, ICASSP 2013)
  - Optimization techniques for recurrent nets
  - Symbolic sequences (music, language)

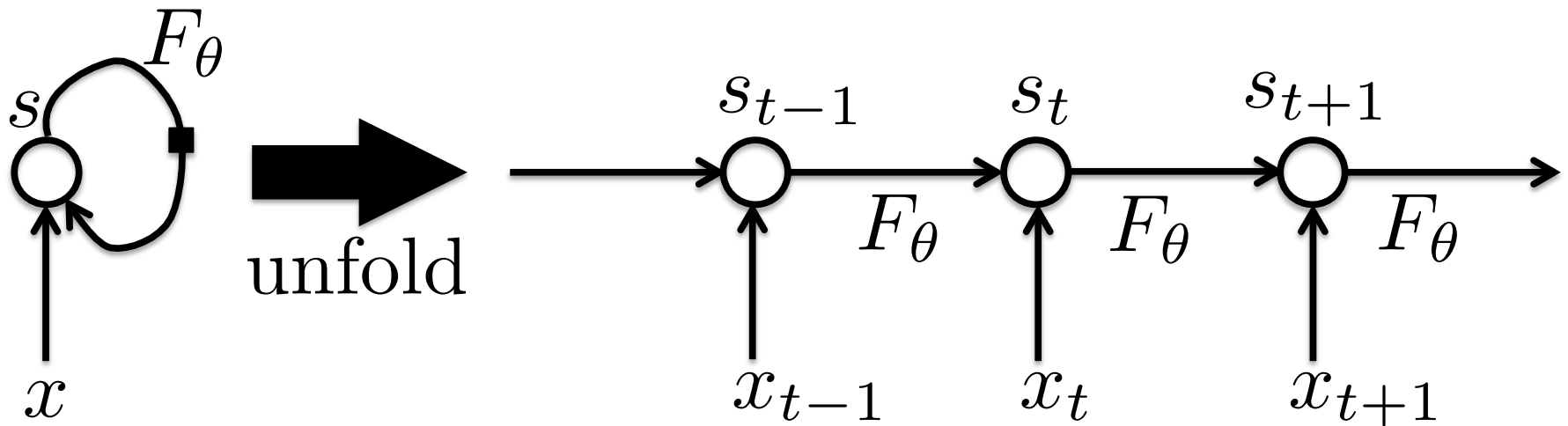
- (Pascanu, Mikolov & Bengio, ICML 2013)
  - Handling longer-term dependencies
  - Symbolic sequences (music, language)



# Recurrent Neural Networks

- Selectively summarize an input sequence in a fixed-size state vector via a recursive update

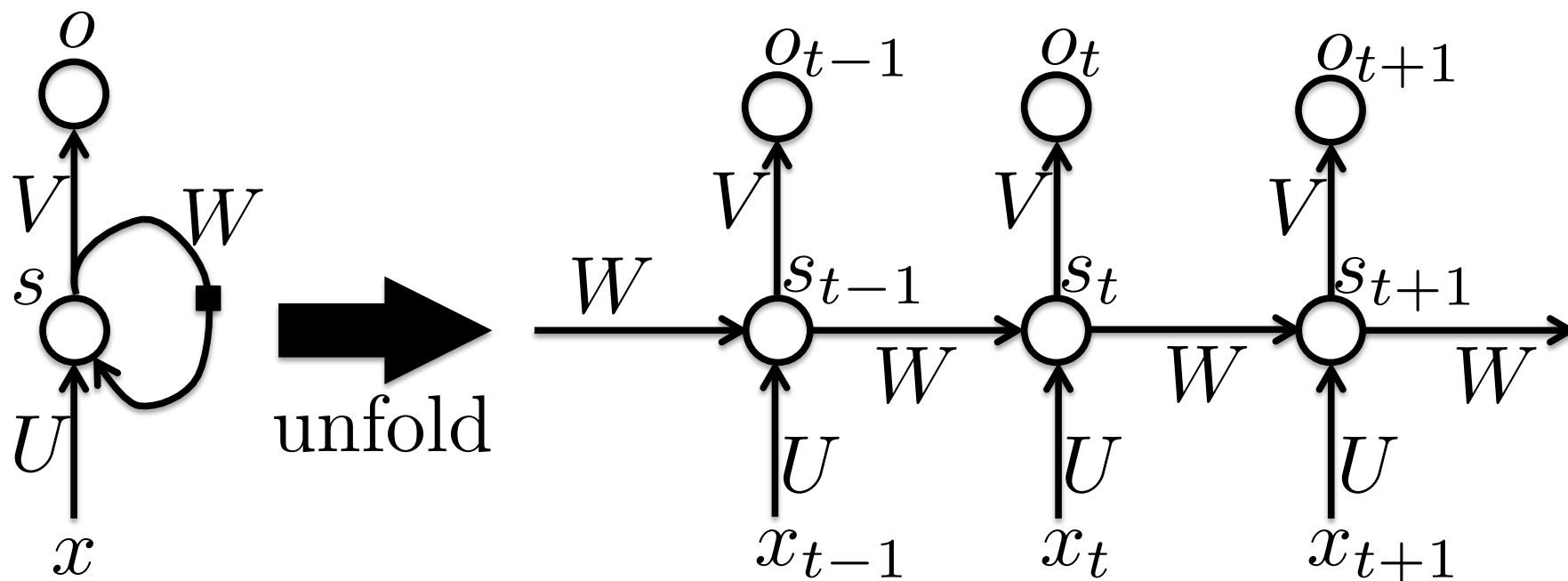
$$s_t = F_\theta(s_{t-1}, x_t)$$



$$s_t = G_t(x_t, x_{t-1}, x_{t-2}, \dots, x_2, x_1)$$

# Recurrent Neural Networks

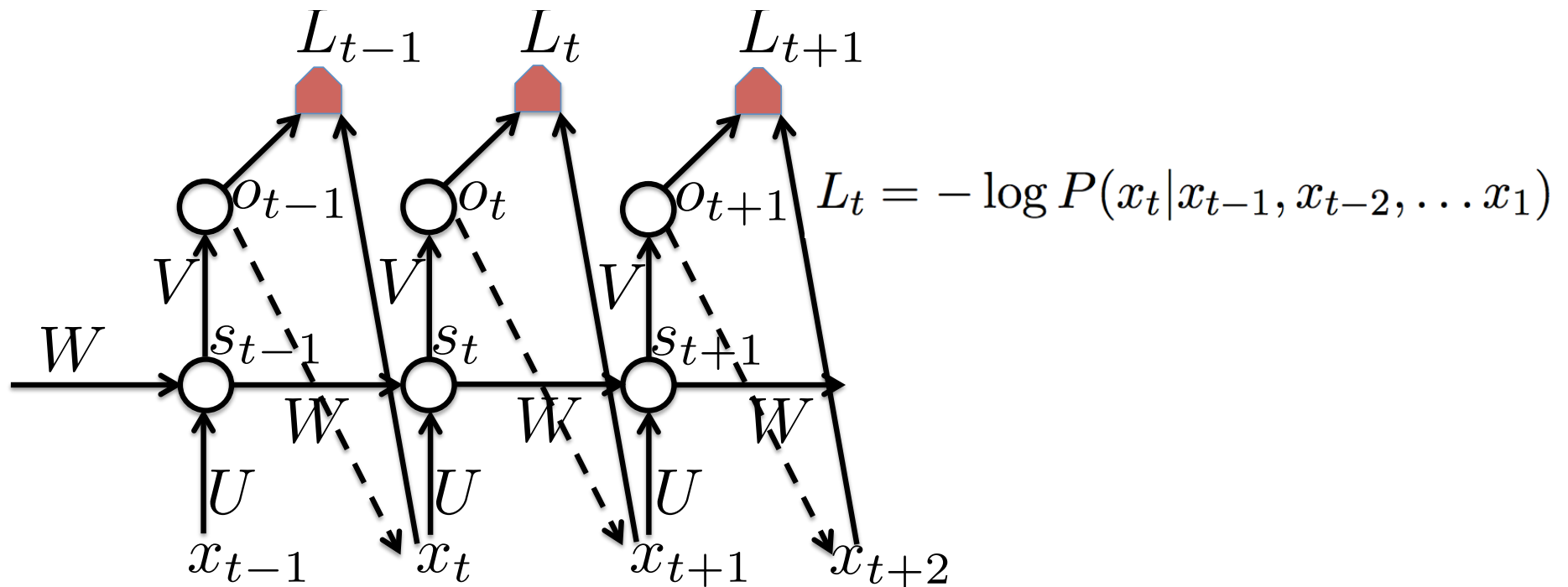
- Can produce an output at each time step: unfolding the graph tells us how to back-prop through time.



# Generative RNNs

- An RNN can represent a fully-connected directed generative model: every variable predicted from all previous ones.

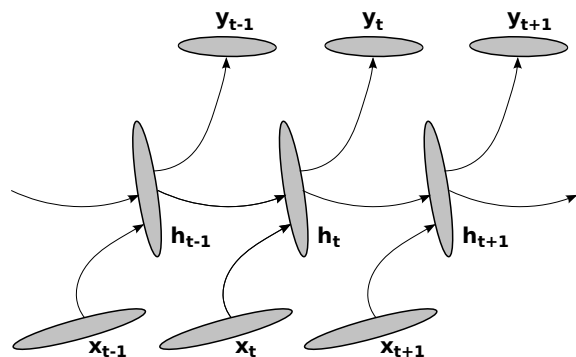
$$P(\mathbf{x}) = P(x_1, \dots, x_T) = \prod_{t=1}^T P(x_t | x_{t-1}, x_{t-2}, \dots, x_1)$$





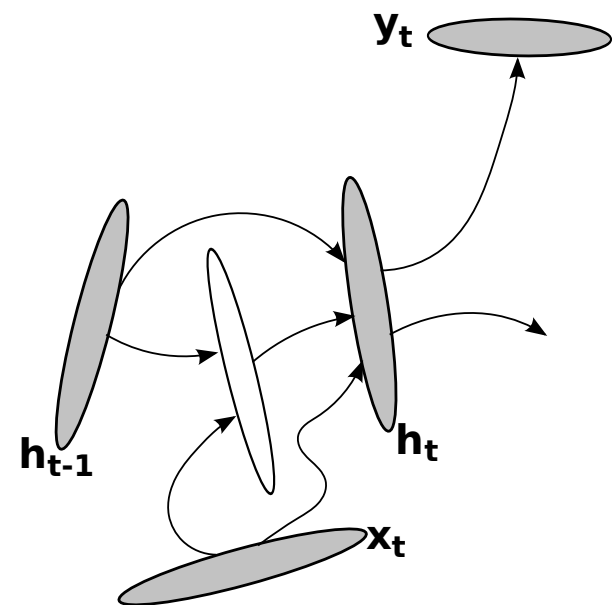
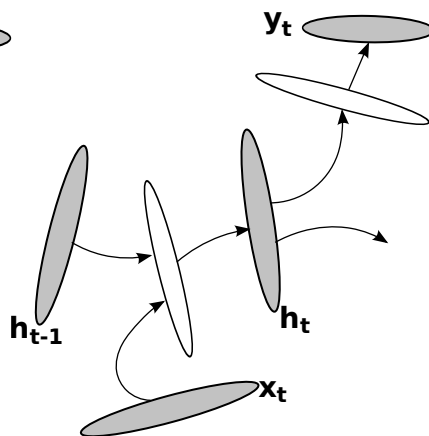
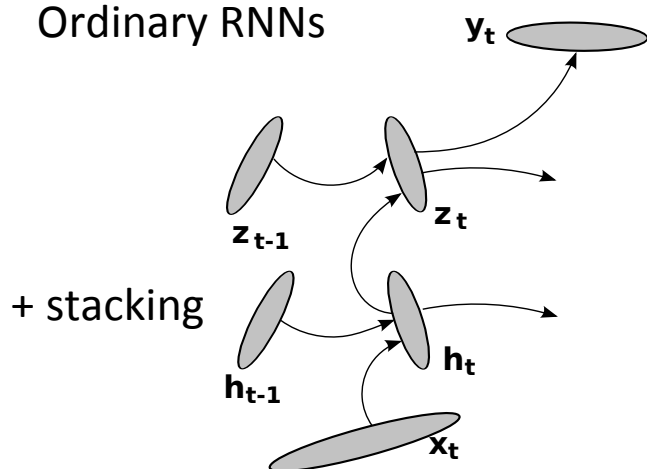
# Increasing the Expressive Power of RNNs with more Depth

- ICLR 2014, *How to construct deep recurrent neural networks*



Ordinary RNNs

+ deep hid-to-out  
+ deep hid-to-hid  
+ deep in-to-hid

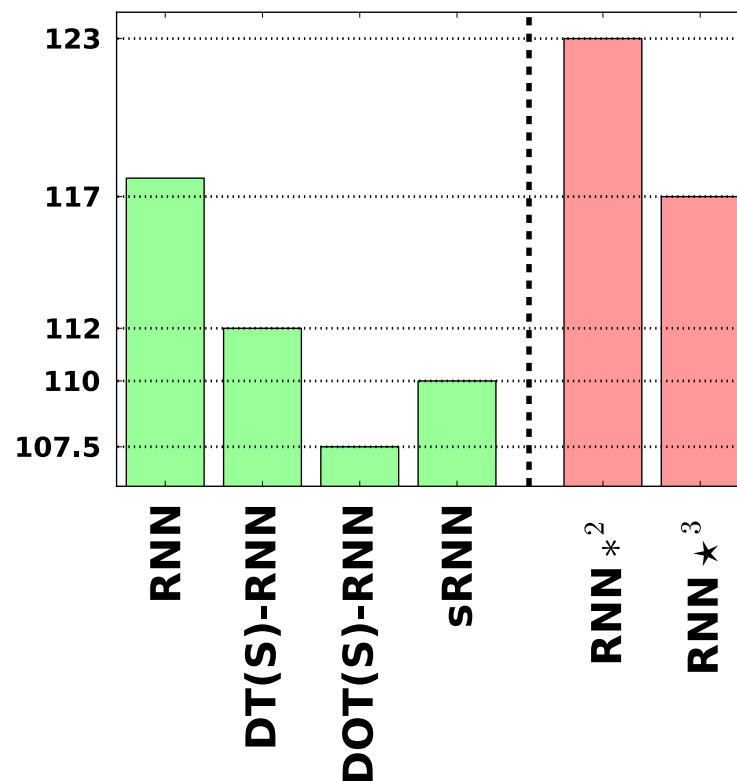
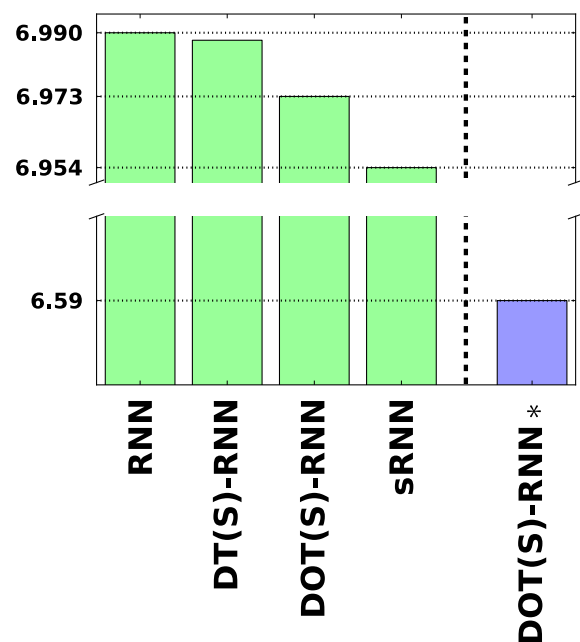


+ skip connections for  
creating shorter paths

# Deep RNN Results

- Language modeling  
(Penn Treebank perplexity)

- Music modeling (Muse, NLL)



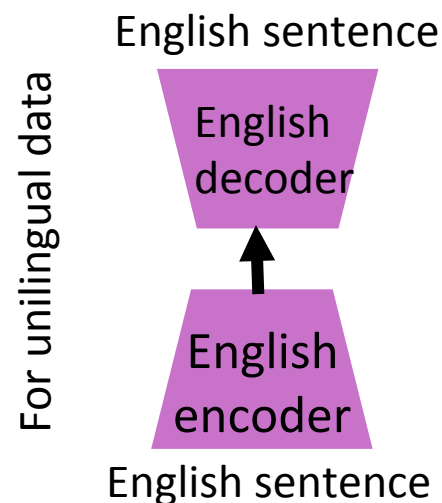
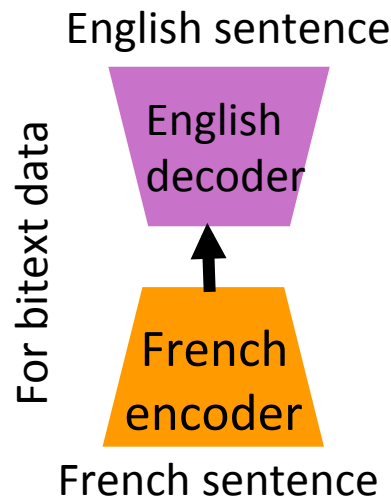
More results in the ICLR 2014 paper

# Already Many NLP Applications of DL

- Language Modeling (Speech Recognition, Machine Translation)
- Acoustic Modeling
- Part-Of-Speech Tagging
- Chunking
- Named Entity Recognition
- Semantic Role Labeling
- Parsing
- Sentiment Analysis
- Paraphrasing
- Question-Answering
- Word-Sense Disambiguation

# Encoder-Decoder Framework for Machine Translation

- One encoder and one decoder per language
- Universal intermediate representation
- $\text{Encode(French)} \rightarrow \text{Decode(English)} = \text{translation model}$
- $\text{Encode(English)} \rightarrow \text{Decode(English)} = \text{language model}$
- Parametrization grows linearly with # languages, not quadratic

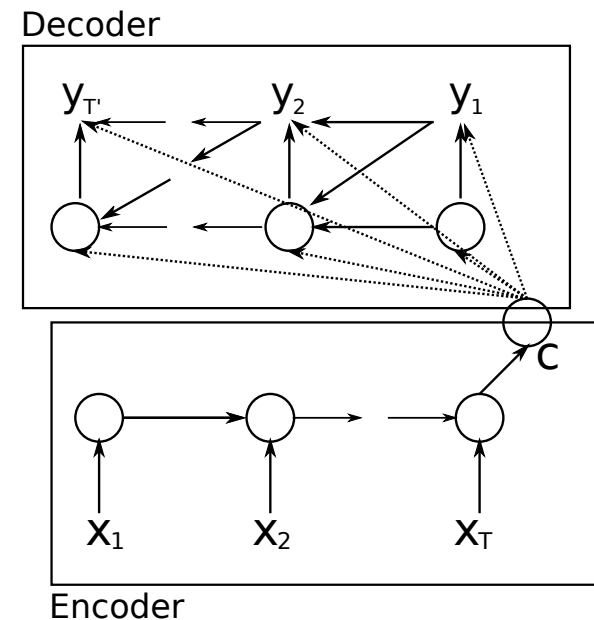


# RNNs for Machine Translation

(Cho, Merrienboer, Gulcehre, Bougares, Schwenk, Bengio; arxiv 2014)

Encoder-decoder framework:

- Encoder = 'summarizing' RNN: word sequence  $\rightarrow$  last-state vector = sequence representation
- Decoder = 'generative' RNN: context  $C \rightarrow$  distribution over word sequences



# RNNs for Machine Translation

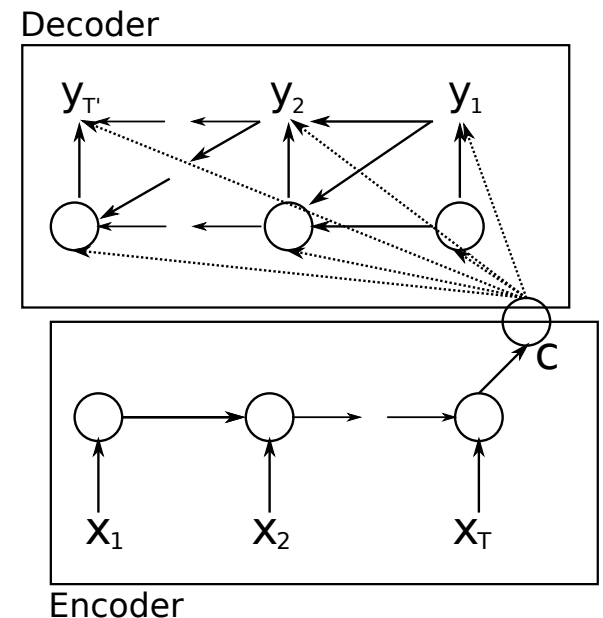
- Decoder = 'generative' RNN: context  $C \rightarrow$  distribution over word sequences

- $P(Y_1..Y_{T'} \mid C) = \prod P(Y_t \mid H_t, C)$

where hidden state  $H_t$  summarizes past seq.

$$H_t = f(H_{t-1}, Y_{t-1}, C) = F(Y_{t-1}, \dots, Y_1, C)$$

- Directed graphical model: ancestral sampling from  $Y_1$  to  $Y_{T'}$ .
- Output sequence can be of different length  $T' \neq T$  not necessarily aligned with input sequence



## RNNs for MT: Results

- English-French WMT'14 task
- Train on both bilingual (supervised) and unilingual (unsupervised)
- Trained on phrases (phrase table), added into log-linear model of MOSES

Models	BLEU	
	dev	test
Baseline	27.63	29.33
CSLM	28.33	29.58
RNN	28.48	29.96
CSLM + RNN	28.60	30.64
CSLM + RNN + WP	28.93	31.18


+1.85 BLEU points



# Practical Considerations



# Deep Learning Tricks of the Trade

- Y. Bengio (2013), “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,  $\theta$  = parameter vector, and  $\epsilon_t$  = learning rate.
- Ordinary gradient descent is a batch method, very slow, **should never be used**. 2<sup>nd</sup> 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  $\epsilon_0$  and  $\tau$ .

- New papers on adaptive learning rates procedures (Schaul 2012, 2013), Adagrad (Duchi et al 2011 ), ADADELTA (Zeiler 2012)

# 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 # of training 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.

# Long-Term Dependencies



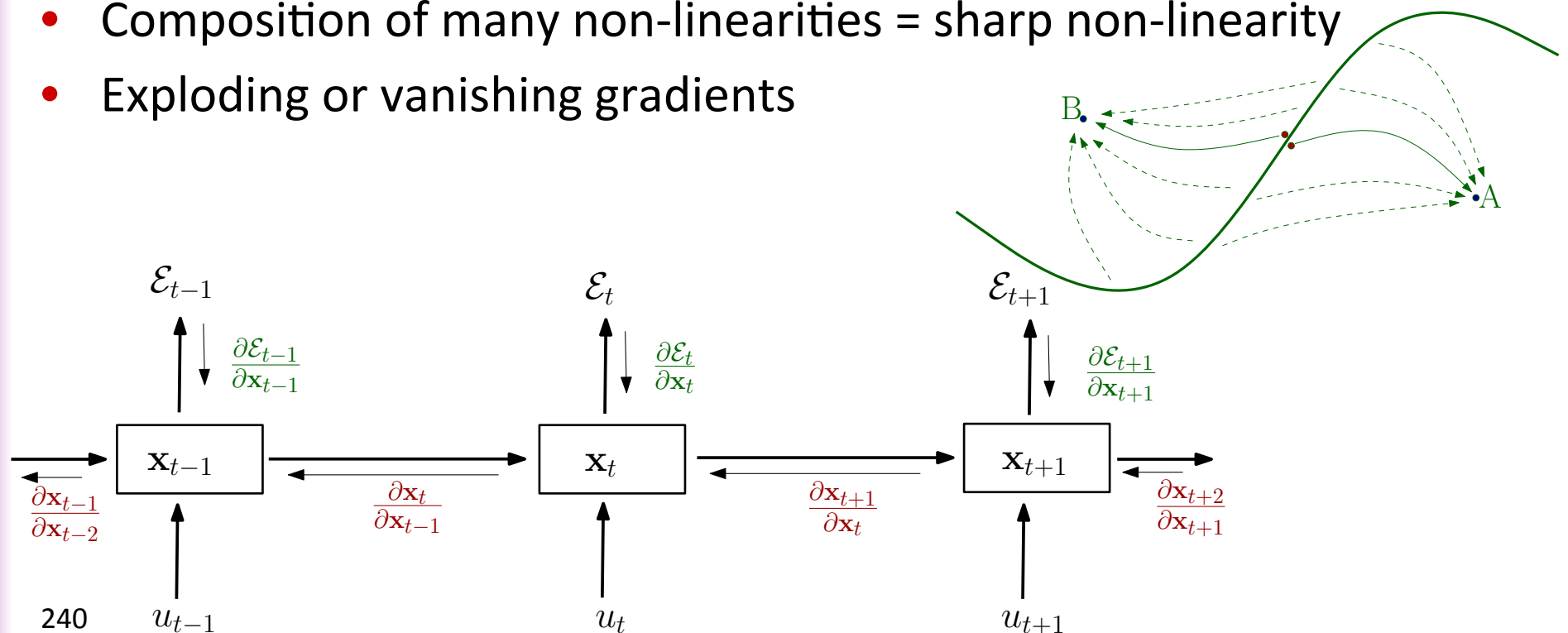
- In very deep networks such as **recurrent networks** (or possibly recursive ones), the gradient is a product of Jacobian matrices, each associated with a step in the forward computation. This can become very small or very large quickly [Bengio et al 1994], and the locality assumption of gradient descent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

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

- Two kinds of problems:
  - sing. values of Jacobians  $> 1 \rightarrow$  gradients explode
  - or sing. values  $< 1 \rightarrow$  gradients shrink & vanish

# The Optimization Challenge in Deep / Recurrent Nets

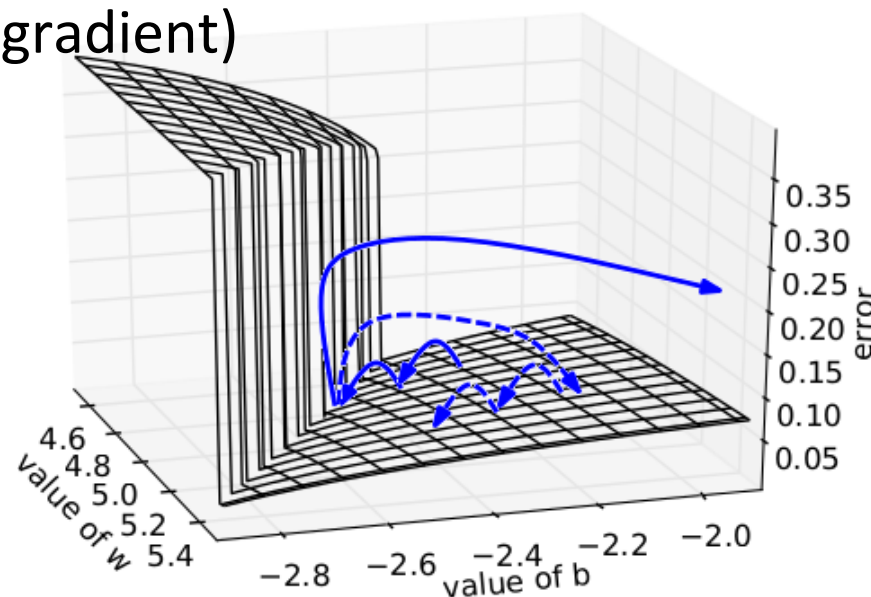
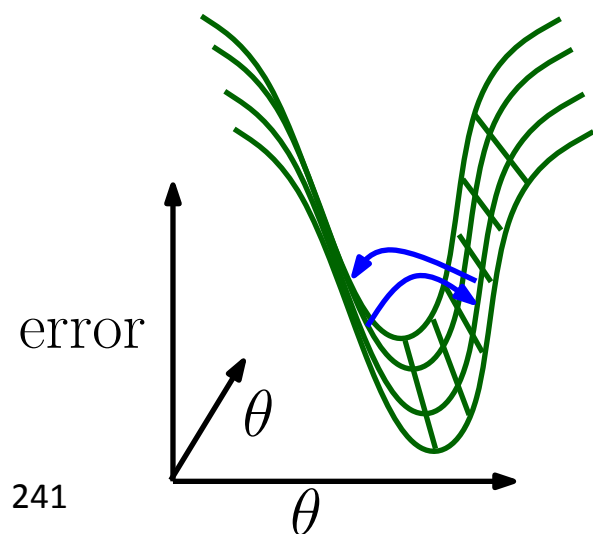
- Higher-level abstractions require highly non-linear transformations to be learned
- Sharp non-linearities are difficult to learn by gradient
- Composition of many non-linearities = sharp non-linearity
- Exploding or vanishing gradients



# RNN Tricks

(Pascanu, Mikolov, Bengio, ICML 2013; Bengio, Boulanger & Pascanu, ICASSP 2013)

- Clipping gradients (avoid exploding gradients)
- Leaky integration (propagate long-term dependencies)
- Momentum (cheap 2<sup>nd</sup> order)
- Initialization (start in right ballpark avoids exploding/vanishing)
- Sparse Gradients (symmetry breaking)
- Gradient propagation regularizer (avoid vanishing gradient)
- LSTM self-loops (avoid vanishing gradient)

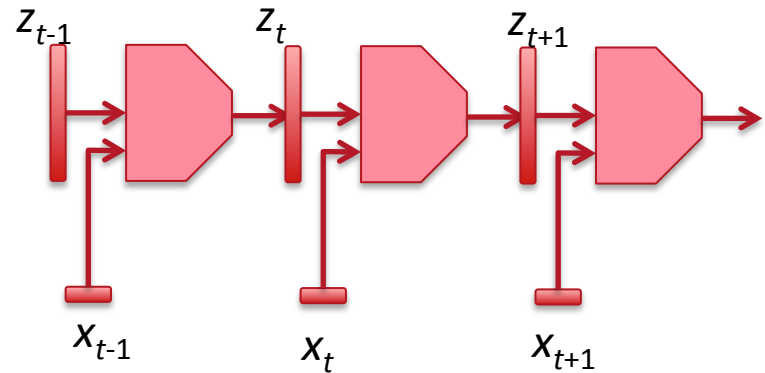


## Gradient Norm Clipping

$\hat{\mathbf{g}} \leftarrow \frac{\partial error}{\partial \theta}$   
**if**  $\|\hat{\mathbf{g}}\| \geq threshold$  **then**  
     $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|} \hat{\mathbf{g}}$   
**end if**



# Long-Term Dependencies and Clipping Trick



Trick first introduced by Mikolov is to clip gradients to a maximum NORM value.



Makes a big difference in Recurrent Nets (Pascanu et al ICML 2013)

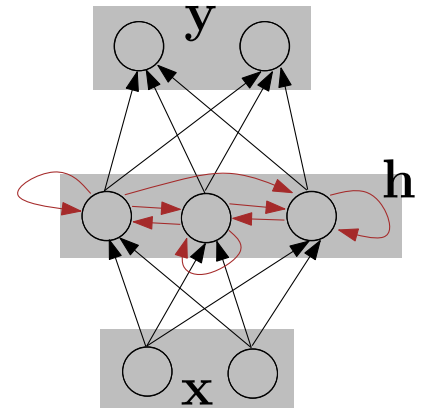
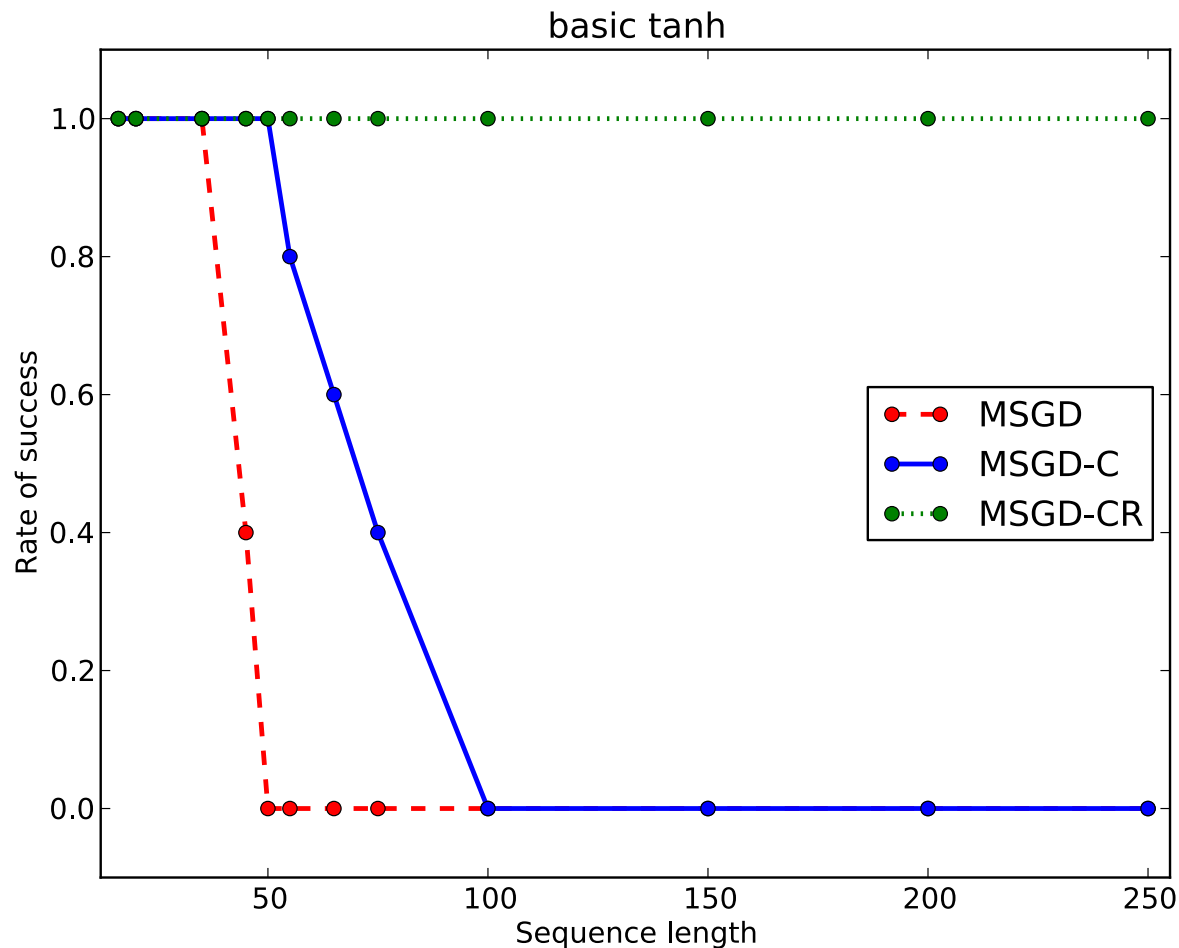
Allows SGD to compete with HF optimization on difficult long-term dependencies tasks. Helped to beat SOTA in text compression, language modeling, speech recognition.

# Temporal Coherence and Scales

- Hints from nature about different explanatory factors:
  - Rapidly changing factors (often noise)
  - Slowly changing (generally more abstract)
  - Different factors at different time scales
- Exploit those **hints** to **disentangle** better!  
(Becker & Hinton 1993, Wiskott & Sejnowski 2002, Hurri & Hyvarinen 2003, Berkes & Wiskott 2005, Mobahi et al 2009, Bergstra & Bengio 2009)
- RNNs working at different time scales  
(Elhihi & Bengio NIPS'1995), (Koutnik et al ICML 2014)

# Combining clipping to avoid gradient explosion and Jacobian regularizer to avoid gradient vanishing

- (Pascanu, Mikolov & Bengio, ICML 2013)



# Normalized Initialization to Achieve Unity-Like Jacobian

Assuming  $f'(\text{act}=0)=1$

To keep information flowing in both direction we would like to have the following properties.

- **Forward-propagation:**

$$\forall(i, i'), Var[z^i] = Var[z^{i'}] \Leftrightarrow \forall i, n_i Var[W^i] = 1$$

- **Back-propagation:**

$$\forall(i, i'), Var\left[\frac{\partial Cost}{\partial s^i}\right] = Var\left[\frac{\partial Cost}{\partial s^{i'}}\right] \Leftrightarrow \forall i, n_{i+1} Var[W^i] = 1$$

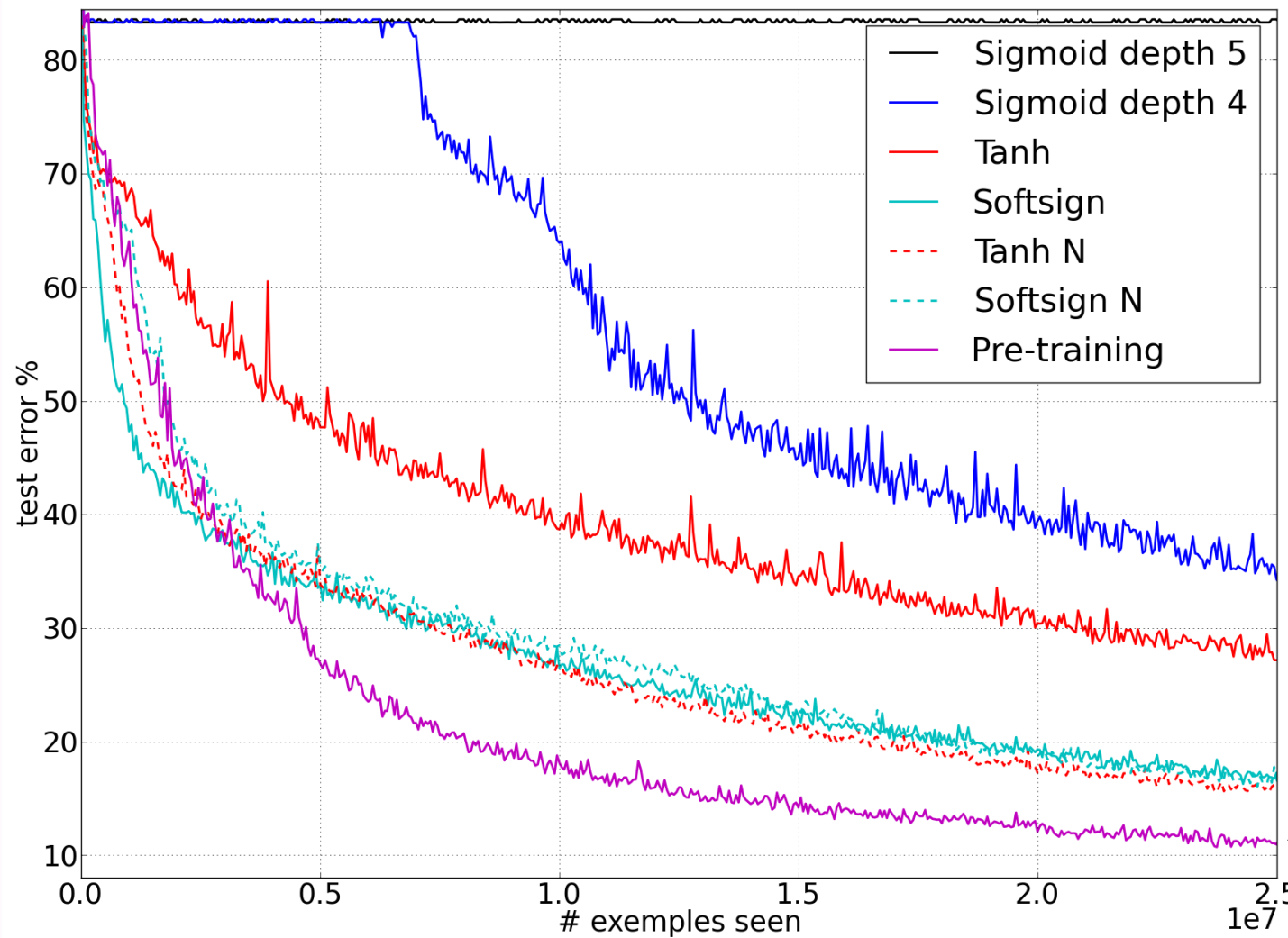
Possible compromise:

$$\forall i, Var[W^i] = \frac{2}{n_i + n_{i+1}} \quad (4)$$

This gives rise to proposed **normalized initialization** procedure:

$$W^i \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_i + n_{i+1}}}, \frac{\sqrt{6}}{\sqrt{n_i + n_{i+1}}}\right] \quad (5)$$

# Normalized Initialization with Variance-Preserving Jacobians

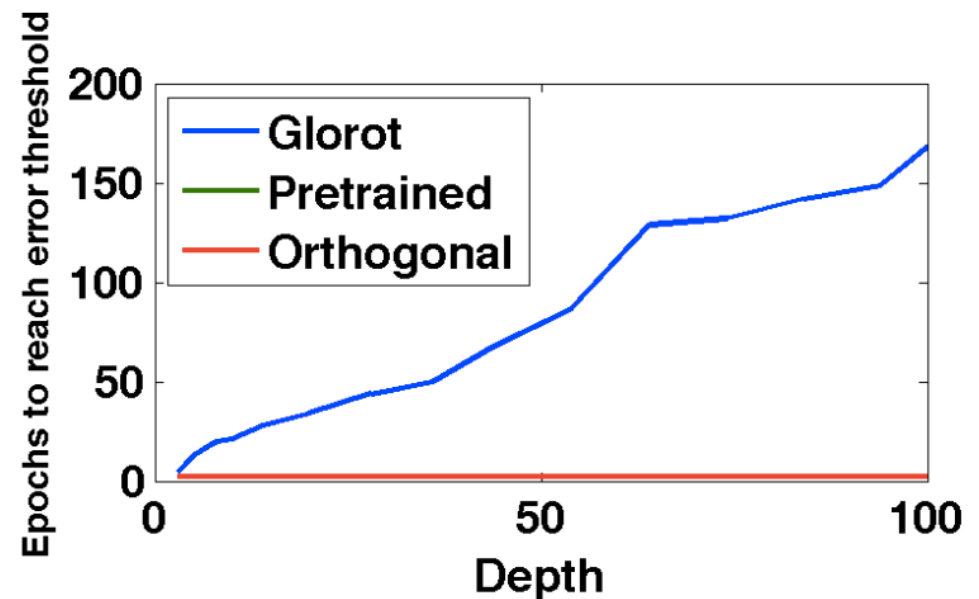


Unsupervised  
pre-training:  
Automatically  
variance-  
preserving!



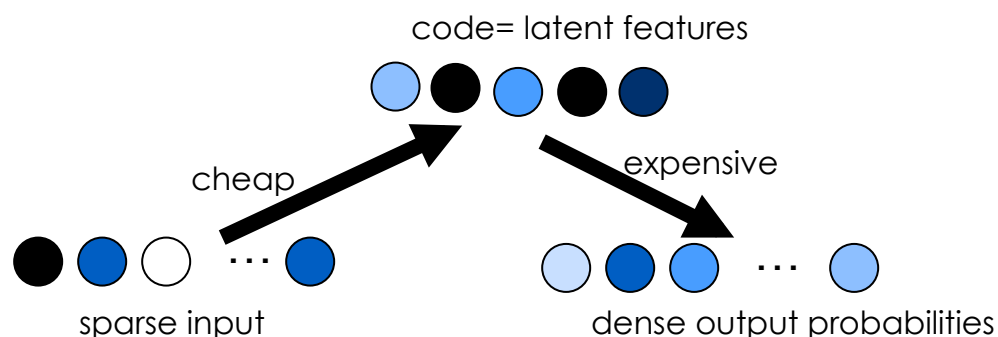
# Orthogonal Initialization Works Even Better

- Auto-encoder pre-training tends to yield orthogonal  $W$
- (Saxe, McClelland & Ganguli ICLR 2014) showed that **very deep** nets initialized with random orthogonal weights are much easier to train
- All singular values = 1



# Handling Large Output Spaces

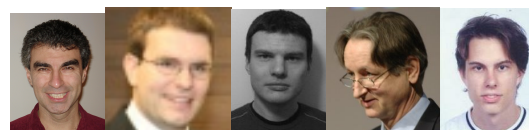
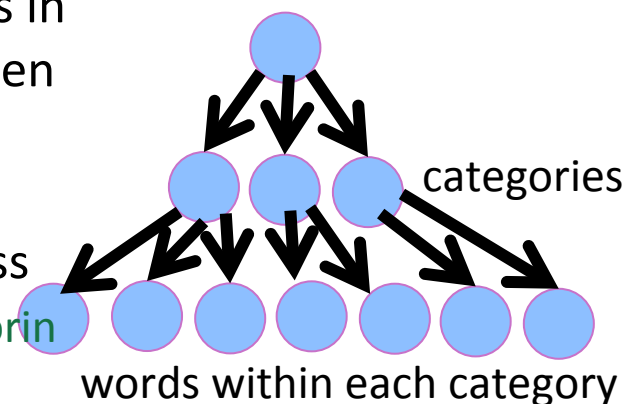
- Auto-encoders and RBMs reconstruct the input, which is sparse and high-dimensional; Language models have a huge output space (1 unit per word).



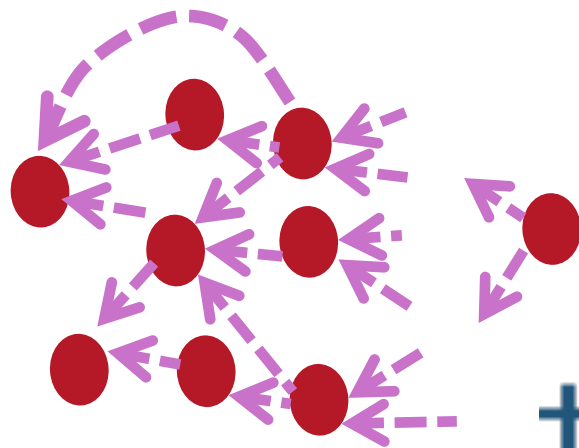
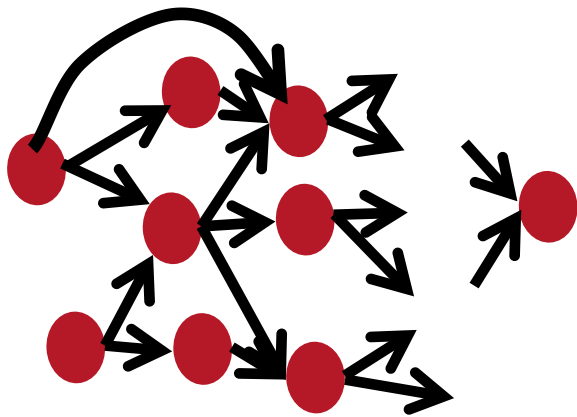
- (Dauphin et al, ICML 2011) Reconstruct the non-zeros in the input, and reconstruct as many randomly chosen zeros, + importance weights



- (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



- Makes it easier to quickly and safely try new models.
- 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)

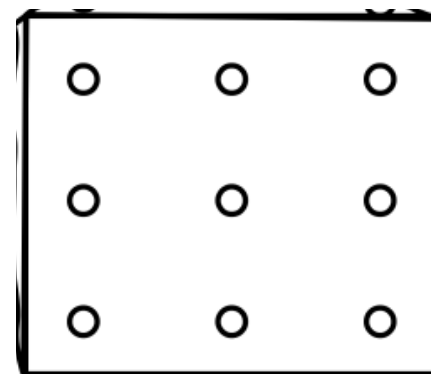


# theano



# Grid Search for Hyper-Parameters

- Discretize hyper-parameter values
- Form cross-product of values across all hyper-parameters: the grid
- Launch a trial training + validation error measurement for each element of the grid
- Can be parallelized on a cluster, but may need to redo failed experiments, until all grid is filled
- <sup>251</sup> Exponential in # of hyper-parameters!



# Examples of hyper-parameters in DL

- Initial learning rate
- Learning rate decrease rate
- Number of layers
- Layer size
- Non-linear activation function
- Output non-linearity
- Output cost function
- Minibatch size
- Skip connections
- Dropout probability
- L1 regularizer, L2 regularizer
- Max weight vector norm
- Pre-training algorithm

Other hyper-parameters:

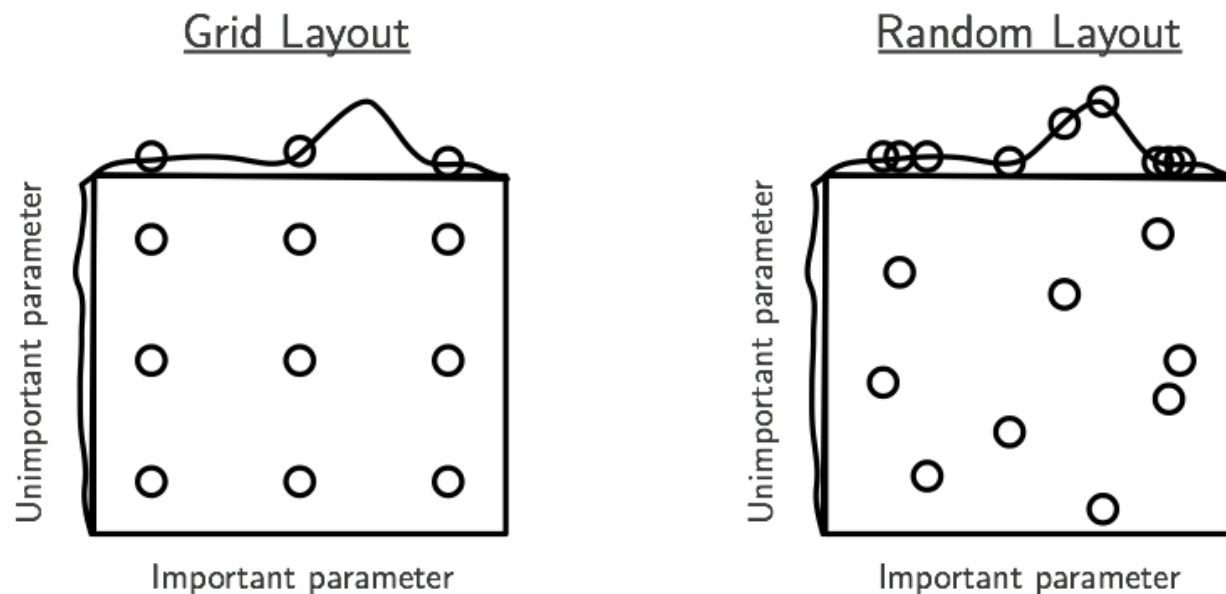
- Pre-training hyper-parameters
- Momentum
- Gradient clipping norm
- Early stopping patience
- Input normalization
- Input dimensionality reduction
- Convolution kernels widths
- Convolutions stride
- Pooling windows sizes
- Pooling strides
- Number of shared layers in multi-task settings
- Output layer regularizer
- Embeddings dimension

# Random Sampling of Hyperparameters

(Bergstra & Bengio 2012)

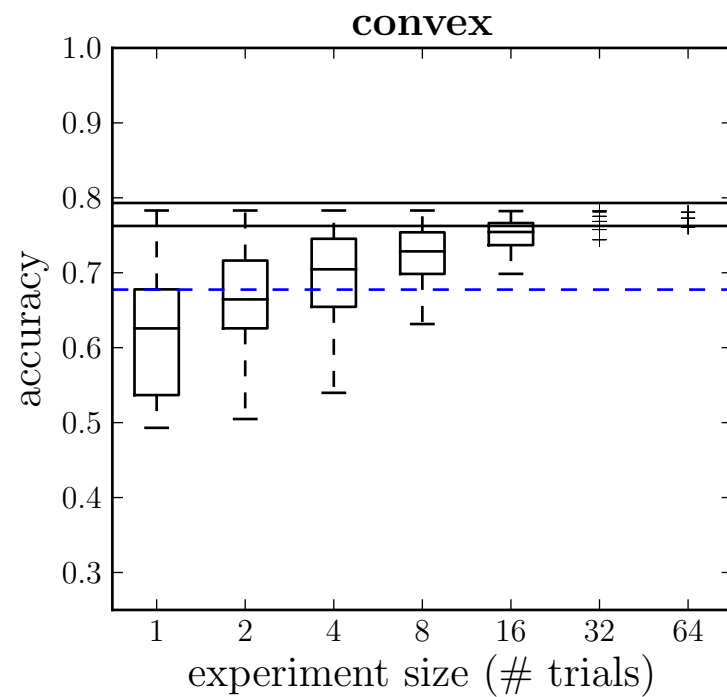
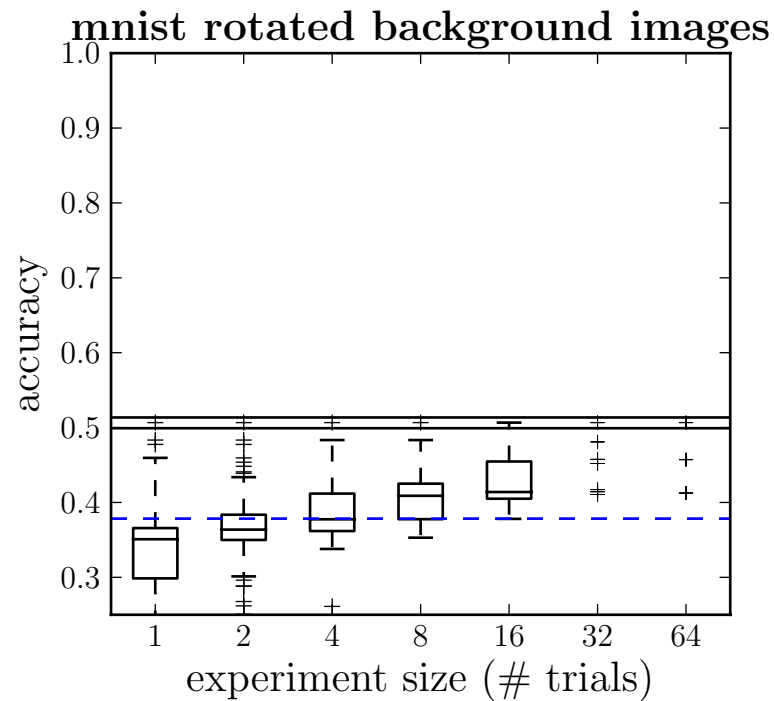


- Random search: simple & efficient
  - Independently sample each HP, e.g.  $\text{l.rate} \sim \exp(\text{U}[\log(.1), \log(.0001)])$
  - Each training trial is iid
  - If a HP is irrelevant grid search is wasteful
  - More convenient: ok to early-stop, continue further, etc.



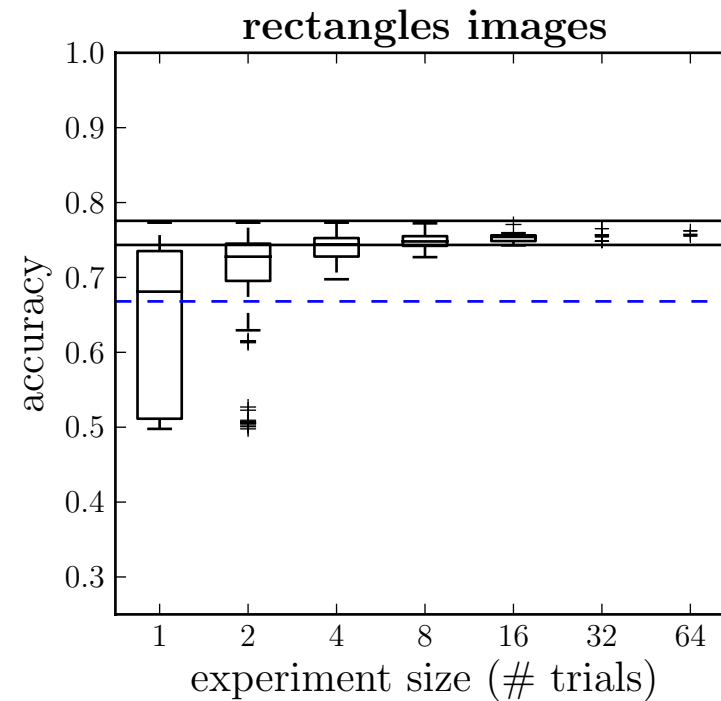
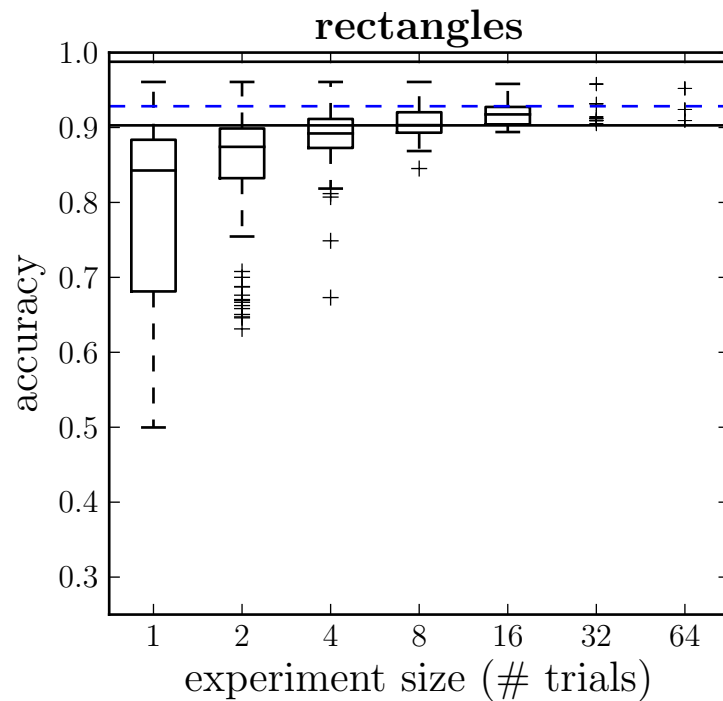
# Random Search Learning Curves

- Blue dotted line = grid search with 100 trials

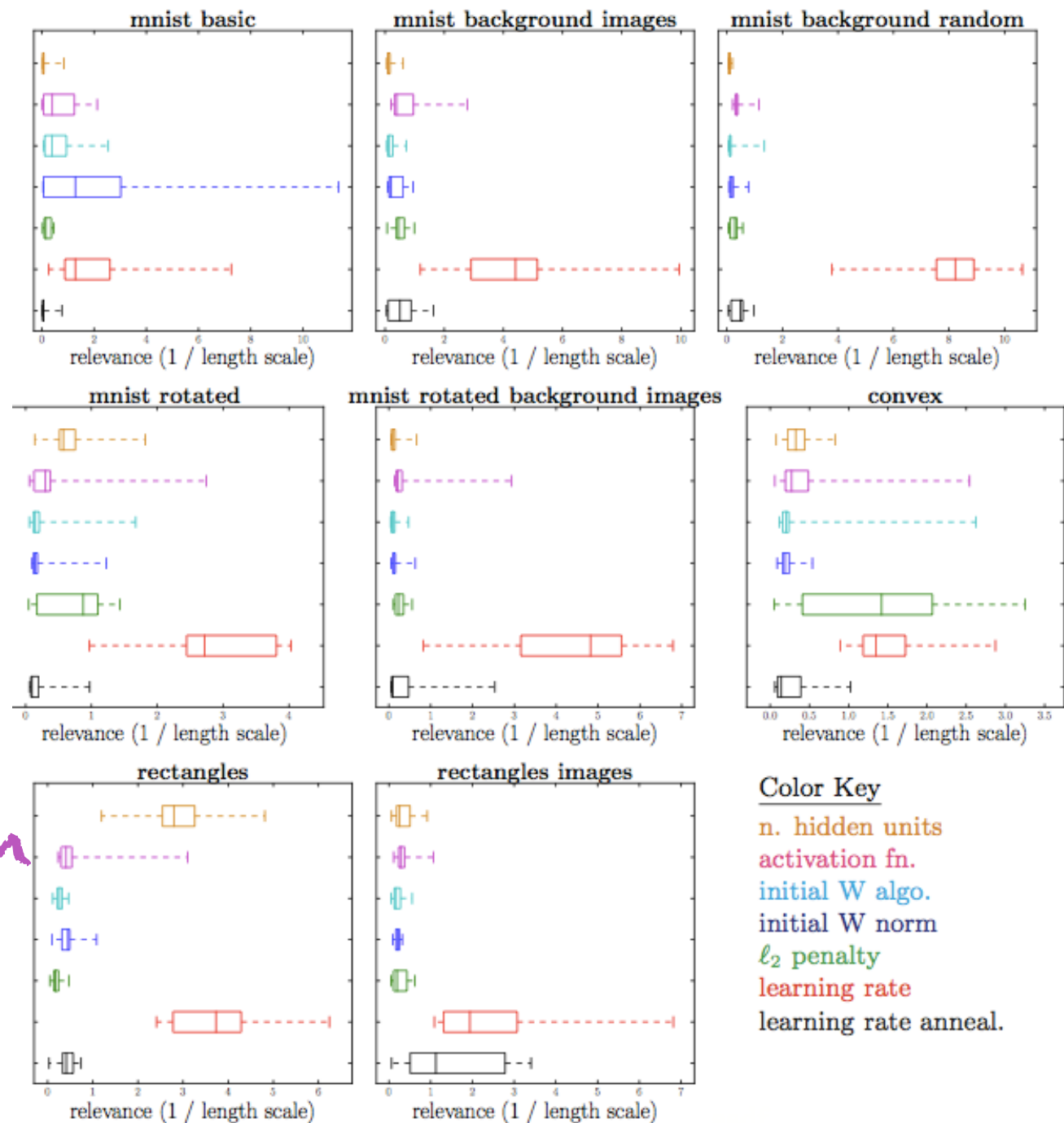


# Random Search Learning Curves

- Blue dotted line = grid search with 100 trials

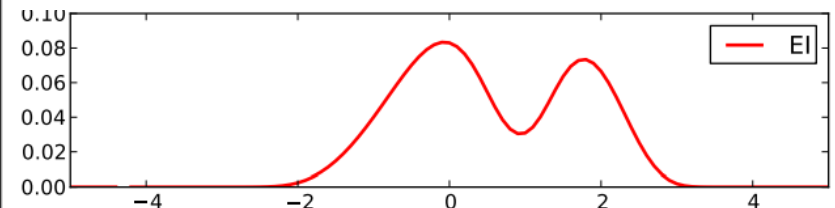
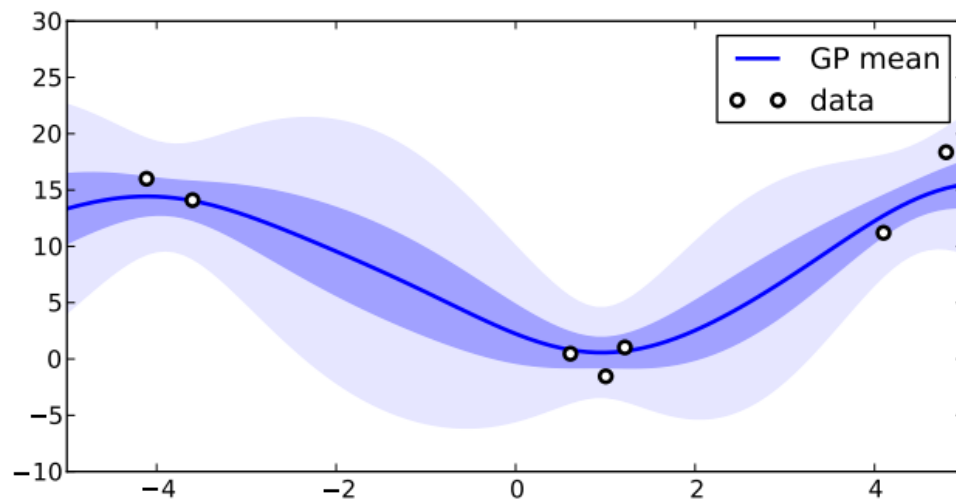


# The Low Effective Dimension of Hyper-Optimization



# Sequential Model-Based Optimization of Hyper-Parameters

- (Hutter et al JAIR 2009; Bergstra et al NIPS 2011; Thornton et al arXiv 2012; Snoek et al NIPS 2012)
- Iterate
  - Estimate  $P(\text{valid. err} \mid \text{hyper-params config } x, D)$
  - choose optimistic  $x$ , e.g.  $\max_x P(\text{valid. err} < \text{current min. err} \mid x)$
  - train with config  $x$ , observe valid. err.  $v$ ,  $D \leftarrow D \cup \{(x, v)\}$



---

Part 4

# Challenges & Questions



# Why is Unsupervised Pre-Training Sometimes Working So Well?

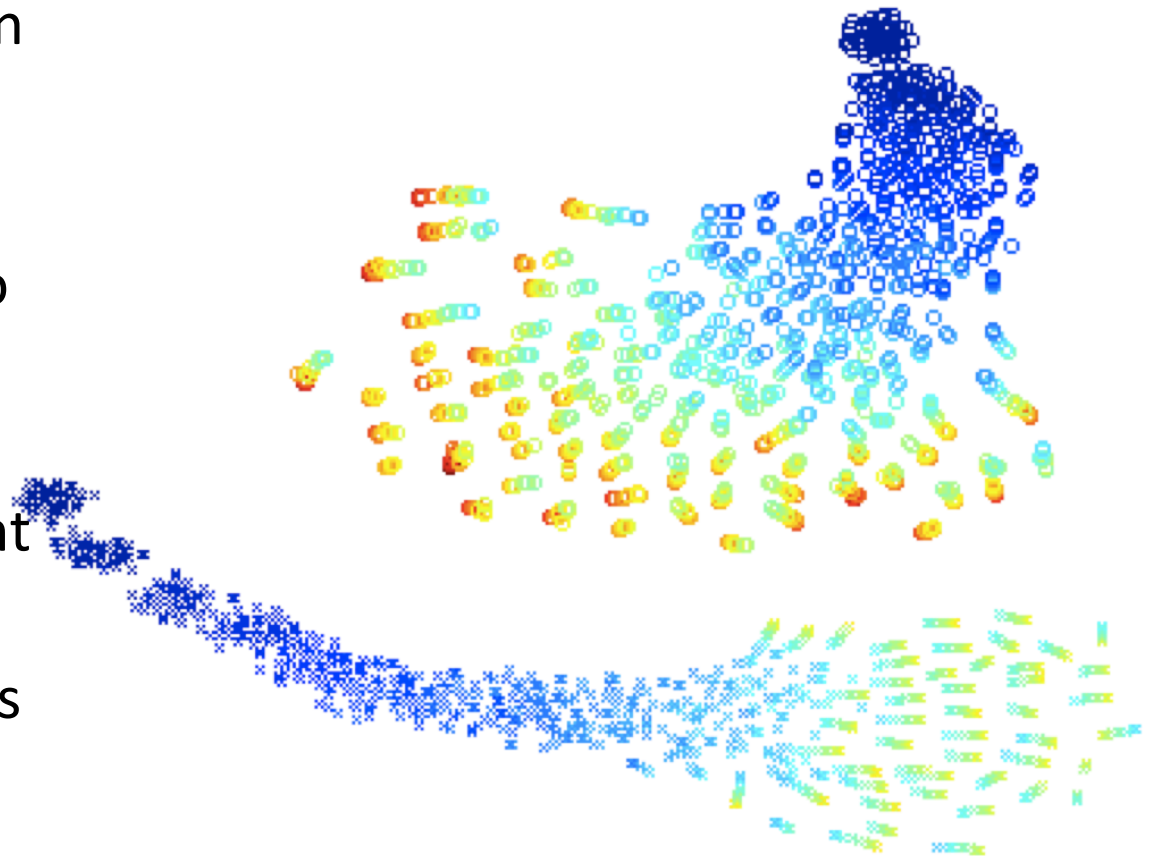
- Regularization hypothesis:
  - Unsupervised component forces model close to  $P(x)$
  - Representations good for  $P(x)$  are good for  $P(y|x)$
- Optimization hypothesis:
  - Unsupervised initialization near better local minimum of  $P(y|x)$
  - Can reach lower local minimum otherwise not achievable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



(Erhan et al JMLR 2010)

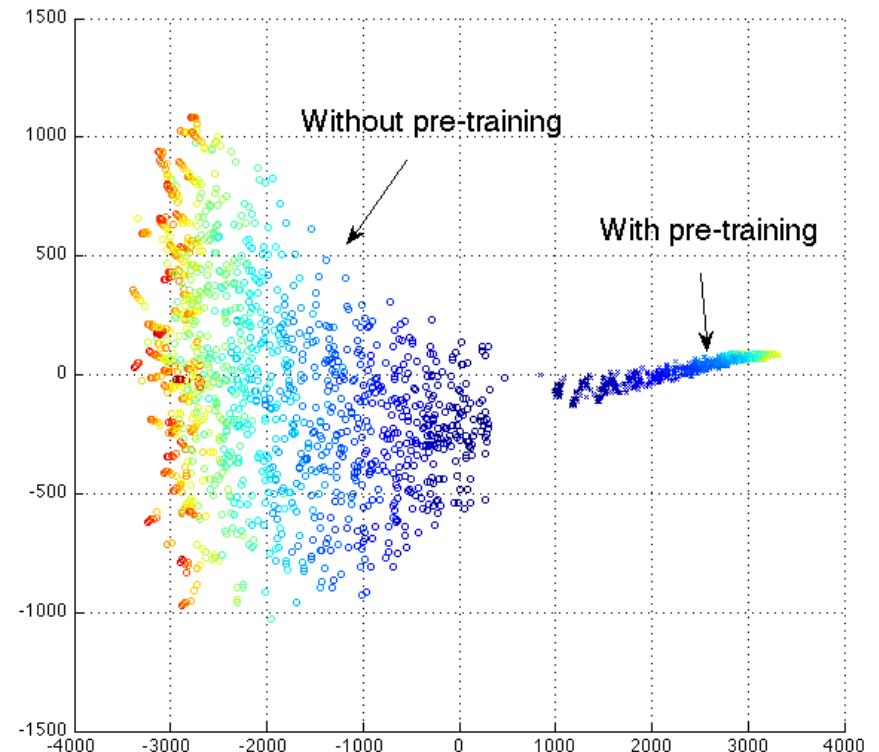
# Learning Trajectories in Function Space

- 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



# Learning Trajectories in Function Space

- Each trajectory converges in different local min.
- With ISOMAP, try to preserve geometry: pretrained nets converge near each other (less variance)
- Good answers = worse than a needle in a haystack (learning dynamics)



# Deep Learning Challenges

(Bengio, arxiv 1305.0445 Deep Learning of representations: Looking forward)

- Computational Scaling
- Optimization & Underfitting
- Intractable Marginalization, Approximate Inference & Sampling
- Disentangling Factors of Variation
- Reasoning & One-Shot Learning of Facts

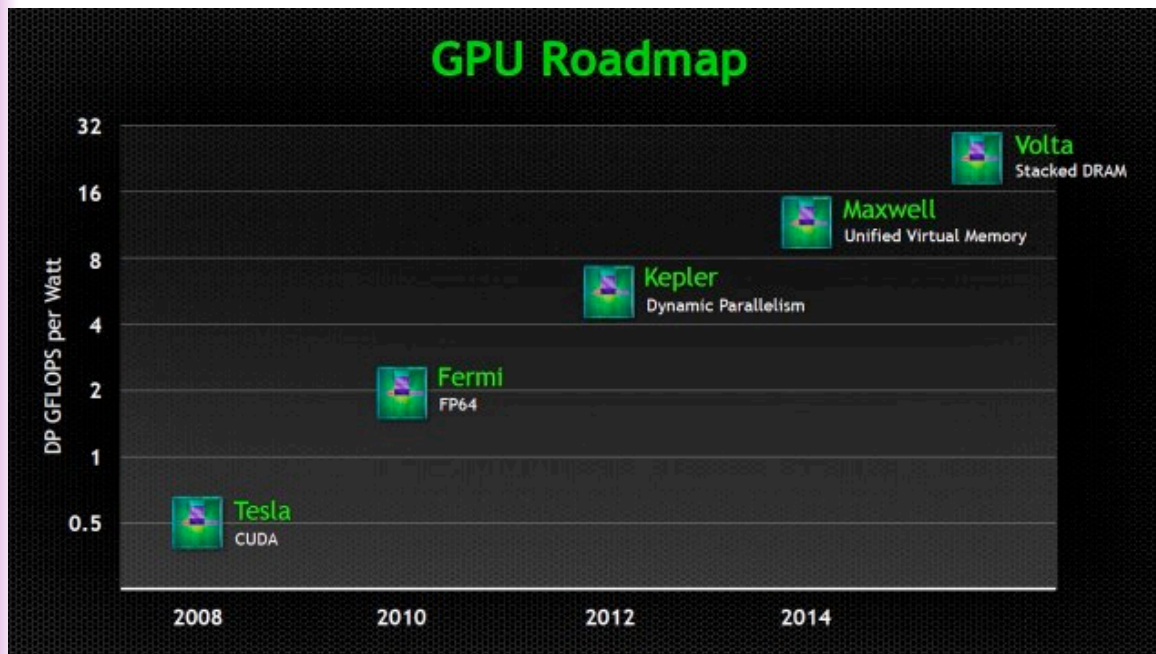
# Deep Learning Challenges

(Bengio, arxiv 1305.0445 Deep Learning of representations: Looking forward)

- Computational Scaling
- Optimization & Underfitting
- Intractable Marginalization, Approximate Inference & Sampling
- Disentangling Factors of Variation
- Reasoning & One-Shot Learning of Facts

# Challenge: Computational Scaling

- Recent breakthroughs in speech, object recognition and NLP hinged on faster computing, GPUs, and large datasets
- A 100-fold speedup is possible without waiting another 10 yrs?
  - Challenge of distributed training
  - Challenge of conditional computation



# Challenge: Computational Scaling

- Recent breakthroughs in speech, object recognition and NLP hinged on faster computing, GPUs, and large datasets
- In speech, vision and NLP applications we tend to find that

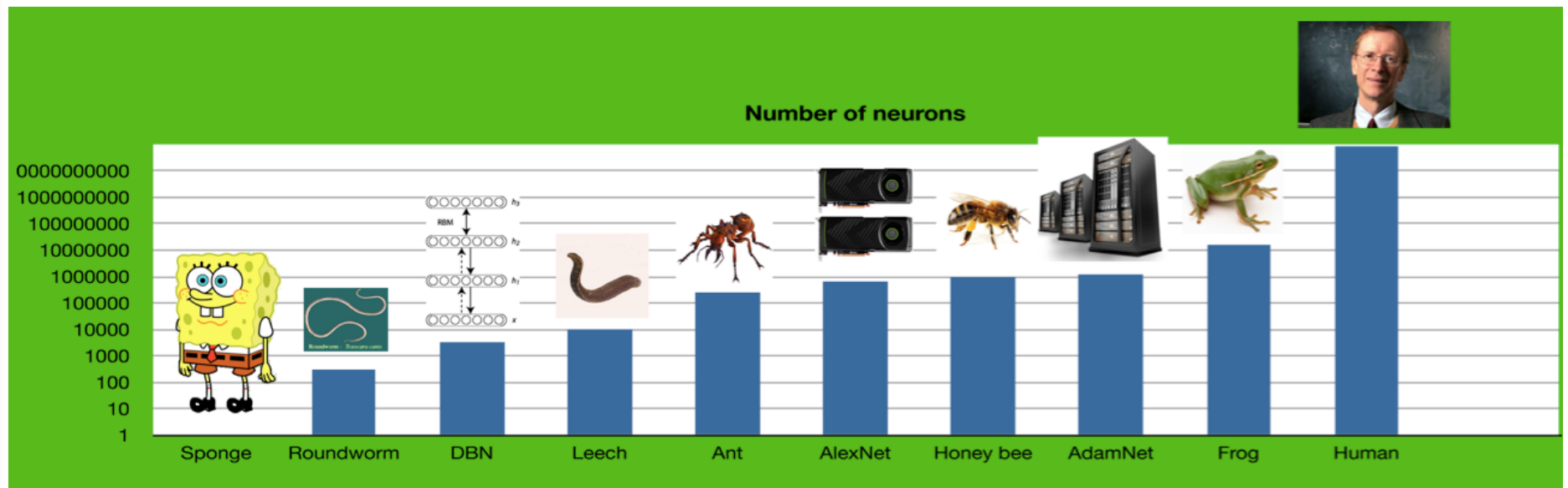
**BIGGER IS BETTER**

Because deep learning is

**EASY TO REGULARIZE** while

it is **MORE DIFFICULT TO AVOID UNDERFITTING**

# We still have a long way to go in raw computational power

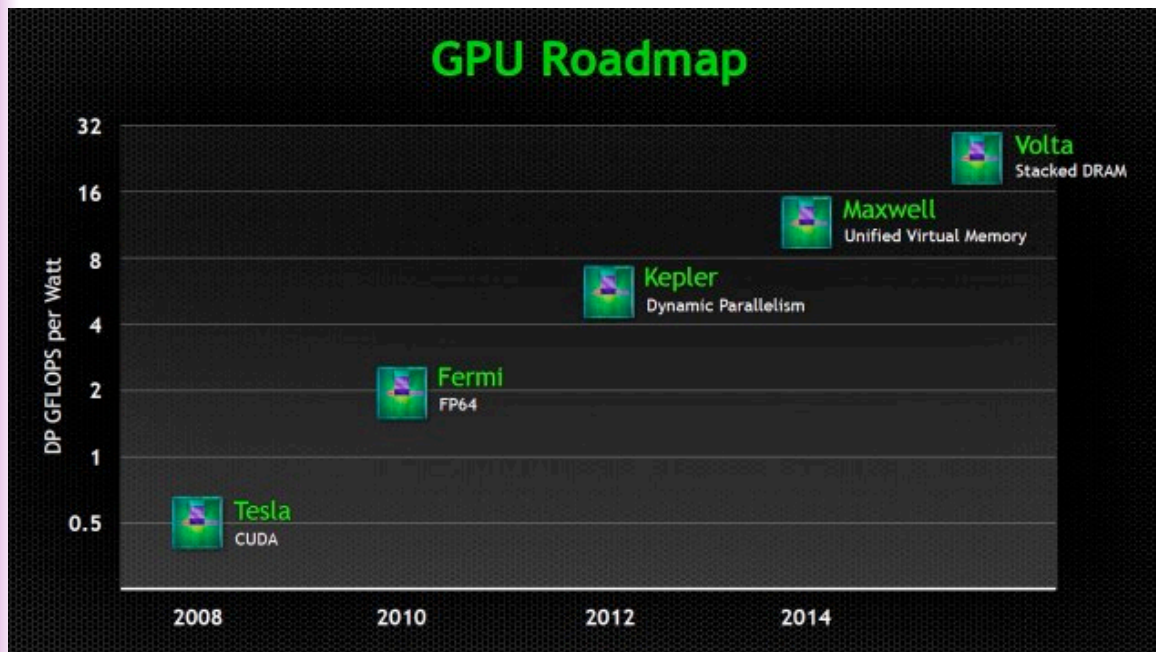




# Hungry for more computing power

- Is a 100-fold or 1000-fold speedup possible without waiting another 10 yrs?
  - Challenge 1: distributed training
  - Challenge 2: conditional computation

Moore's law on single cores has saturated: growth now comes from **parallelization**

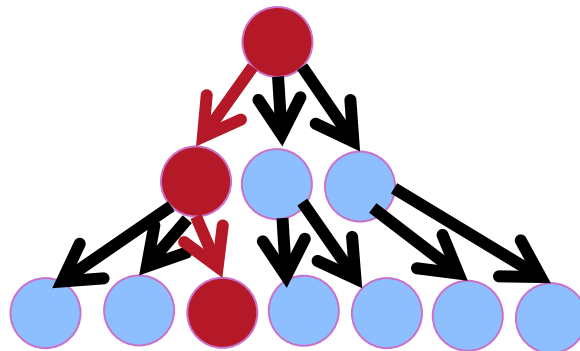


# Distributed Training

- Minibatches
- Large minibatches + 2<sup>nd</sup> order & natural gradient methods
- Asynchronous SGD (Bengio et al 2003, Le et al ICML 2012, Dean et al NIPS 2012)
  - Bottleneck: sharing weights/updates among nodes, to avoid node-models to move too far from each other
- Ideas forward:
  - Low-resolution sharing only where needed
  - Specialized conditional computation (each computer specializes in updates to some cluster of gated experts, and prefers examples which trigger these experts)

100

- # NUMBER OF COMPUTATIONS / NUMBER OF PARAMETERS



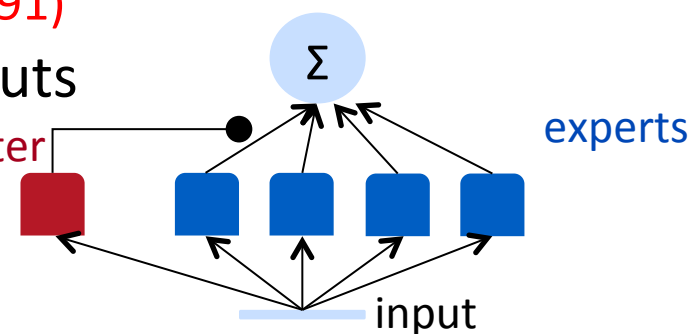
# Conditional Computation: only visit a small fraction of parameters / example

- Regular **mixture of experts** (Jacobs et al 1991)

Output = weighted sum of experts outputs

Gater partitions input space, chooses which expert to listen in each region.

Gater softmax output = weights



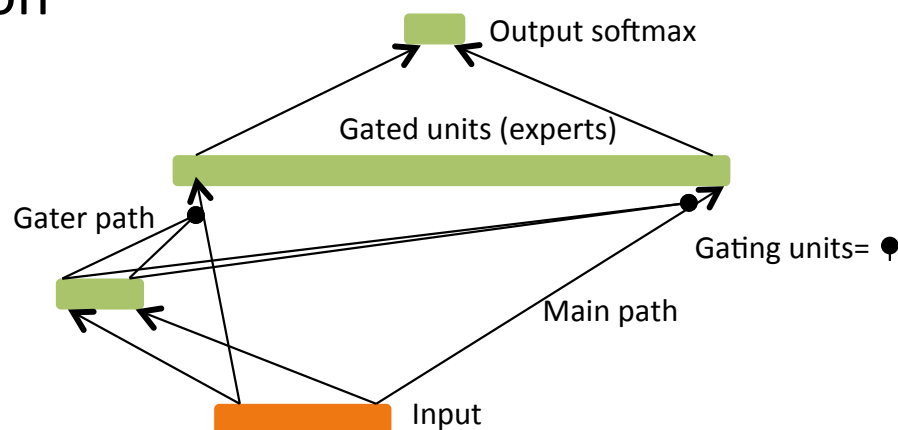
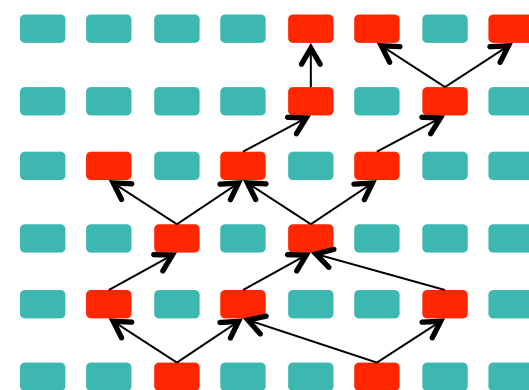
- No computational benefit, but easier optimization (each expert specializes in its gater-assigned region)

- **Hard mixtures of experts** (Collobert, Bengio & Bengio 2002)

- Gater takes a hard decision
- No benefit a training time (need to run all experts to tell gater which one it should have chosen)
- $O(K)$  speedup at test time if  $K$  experts


# Conditional Computation: only visit a small fraction of parameters / example

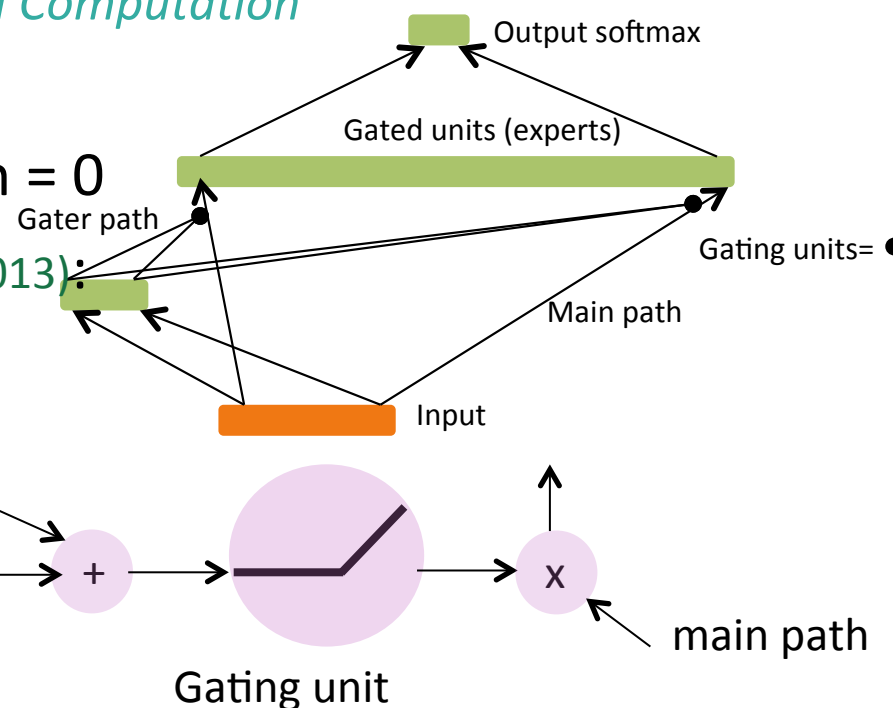
- Conditional computation for deep nets: sparse distributed gaters selecting combinatorial subsets of a deep net
- Challenges:
  - Credit assignment for hard decisions
  - Gated architectures exploration
- (Bengio, Leonard, Courville 2013):  
*Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation*



# Credit Assignment for Discrete Actions

(Bengio, Leonard, Courville 2013): *Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation*

- Gating units take a hard decision
- Gradient through discrete function = 0
- Solutions in (Bengio, Leonard, Courville 2013):
  - Heuristic back-prop (straight through estimator), also (Gregor et al ICML 2014).
  - Noisy rectifier: 
  - Smooth times Stochastic bvp with  $b \sim \text{Bin}(\nu p)$
  - REINFORCE with variance reduction baseline, i.e., RL, i.e. correlate with loss, no back-prop for gaters
- Another option: train a stochastic credit-assignment machine by Reweighted Wake-Sleep (Bornschein & Bengio 2014)

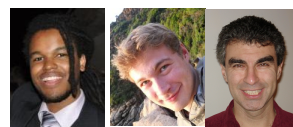
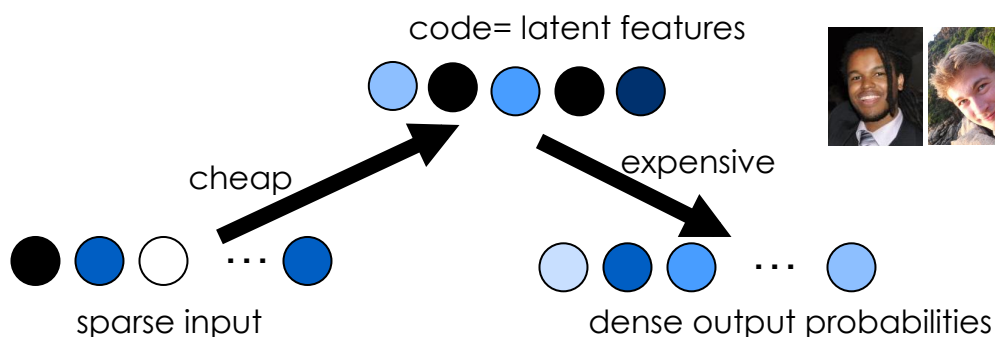


# Conditional Computation on the Output Layer

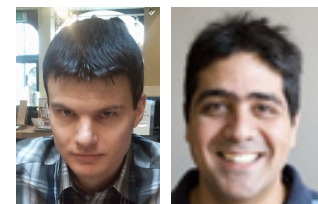
- When computing the loss  $L(f(x), y)$ , we can exploit the knowledge of  $y$  to make the computation of the loss NOT HAVE TO COMPUTE ALL THE PARAMETERS involved in  $f(x)$ .
- Example 1:  $-\log P(y|x)$  can be decomposed in a tree structure over the classes  $y$ , into super-(super-)categories
- Example 2: a sampling approximation of  $L(f(x), y)$  can be computed that is much cheaper

# Handling Large Output Spaces

- Auto-encoders and RBMs reconstruct the input, which is sparse and high-dimensional; Language models have a huge output space (1 unit per word).



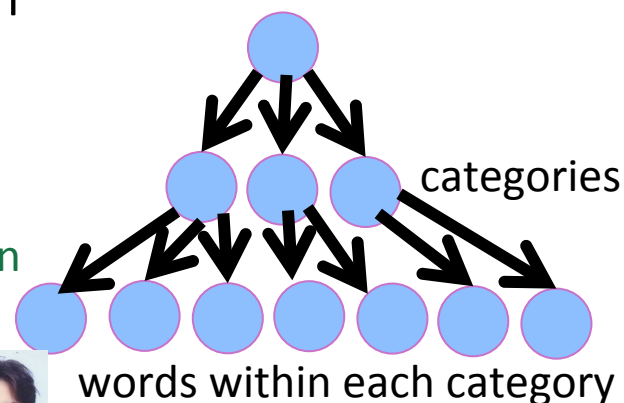
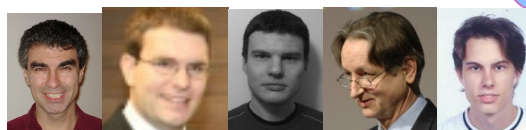
Alternatives to likelihood not requiring the compute the normalization constant, e.g. NCE (Mnih&Kavukcuoglu NIPS 2013)



- (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)





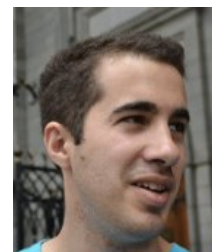
# Exploiting Sparsity

- If  $x$  is sparse, computing  $Wx$  only needs to touch the columns associated with non-zero  $x_i$
- Unfortunately, it is more difficult to exploit sparsity on GPUs, especially when the pattern of sparsity is not the same between examples of the same mini-batch
- Implementations using the sparse matrix multiplications with CUDA can be 100x slower than their dense counterparts (on a per-multiply-add basis) while their CPU equivalents can be 10x slower than their dense counterparts.

# Block-Wise Sparsity

If the sparsity pattern is constrained to be block-wise, with blocks of 16 or 32 (the GPU block size) then one can efficiently handle sparse activations and inputs.

(Leonard et al 2014)



Preliminary experiments on a large neural language model (Billion words dataset, 800k output vocabulary):

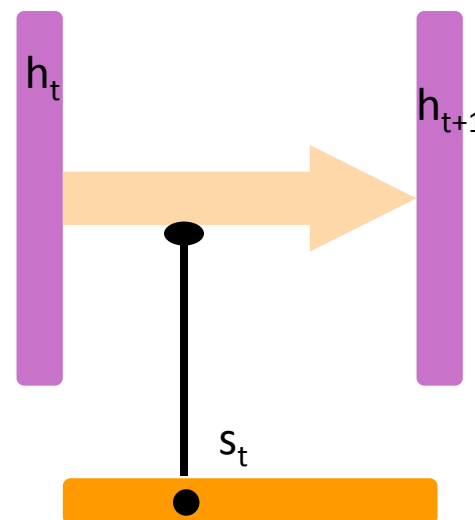


- 50x speedup against same-size dense
- 5x loss compared against same #multiply-add dense

## Sparse inputs + 3-way connections

- Much larger payoff can be obtained when using 3-way connections if the input is very sparse e.g. one-hot code for characters (*Sutskever et al ICML 2011*)
- E.g. Each input symbol  $s_t$  selects a different recurrent weight matrix of an RNN

$$h_{t+1} = \tanh(b + W_{s_t} h_t)$$



# Exponentially Exploding the #Parameters for fixed Computation

(Cho & Bengio 2014)

To drastically increase the ratio of parameters to computation, binarize the pattern of activations of a layer to select up to  $2^k$  weight matrices ( $n \times m$ ) for computing the next layer.

Gater:  $k$  of the  $n$  units

Memory = parameters:  $2^k n m$

Computation:  $n m$

Many variants are possible (may use different  $k$  bits for each hidden unit, may add prefix-indexed matrices, etc.)

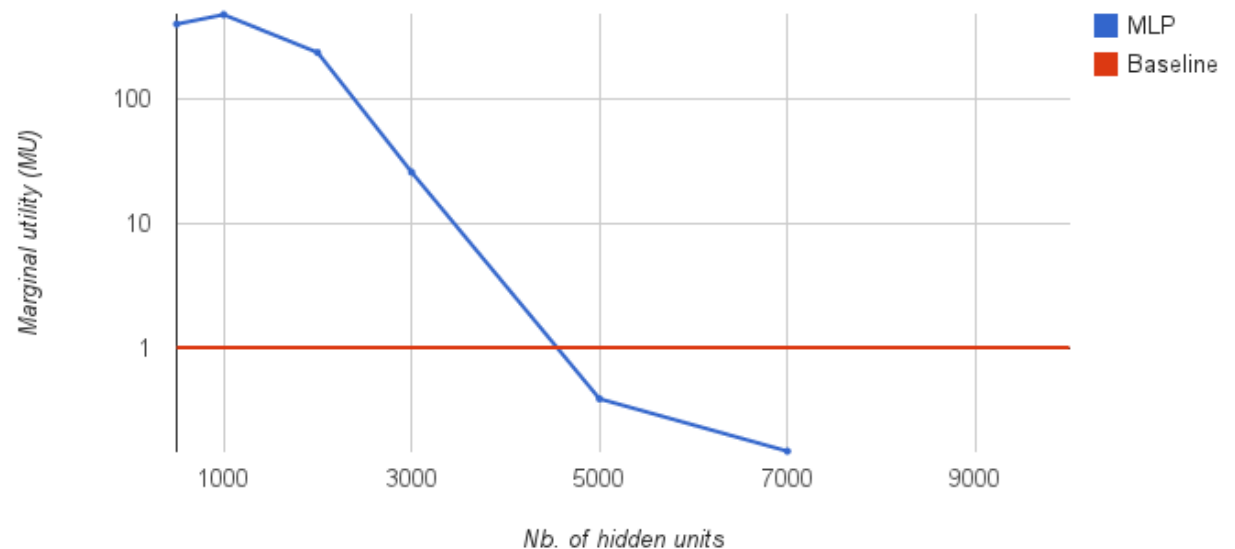
# Deep Learning Challenges

(Bengio, arxiv 1305.0445 Deep Learning of representations: Looking forward)

- Computational Scaling
- Optimization & Underfitting
- Intractable Marginalization, Approximate Inference & Sampling
- Disentangling Factors of Variation
- Reasoning & One-Shot Learning of Facts

# Optimization & Underfitting

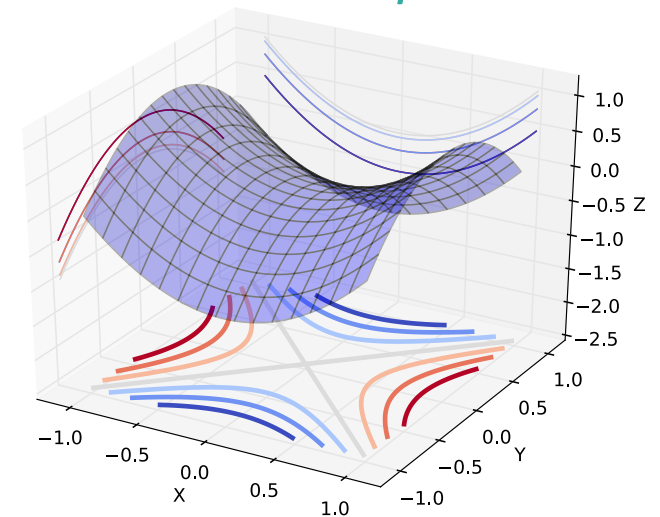
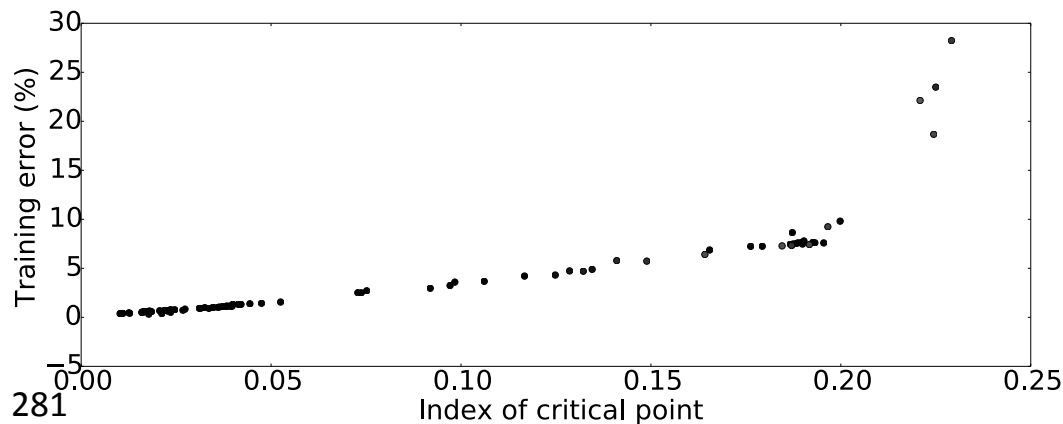
- On large datasets, major obstacle is underfitting
- **Marginal utility** of wider MLPs decreases quickly below memorization baseline



- Current limitations: local minima, ill-conditioning or else?

# Saddle Points, not Local Minima

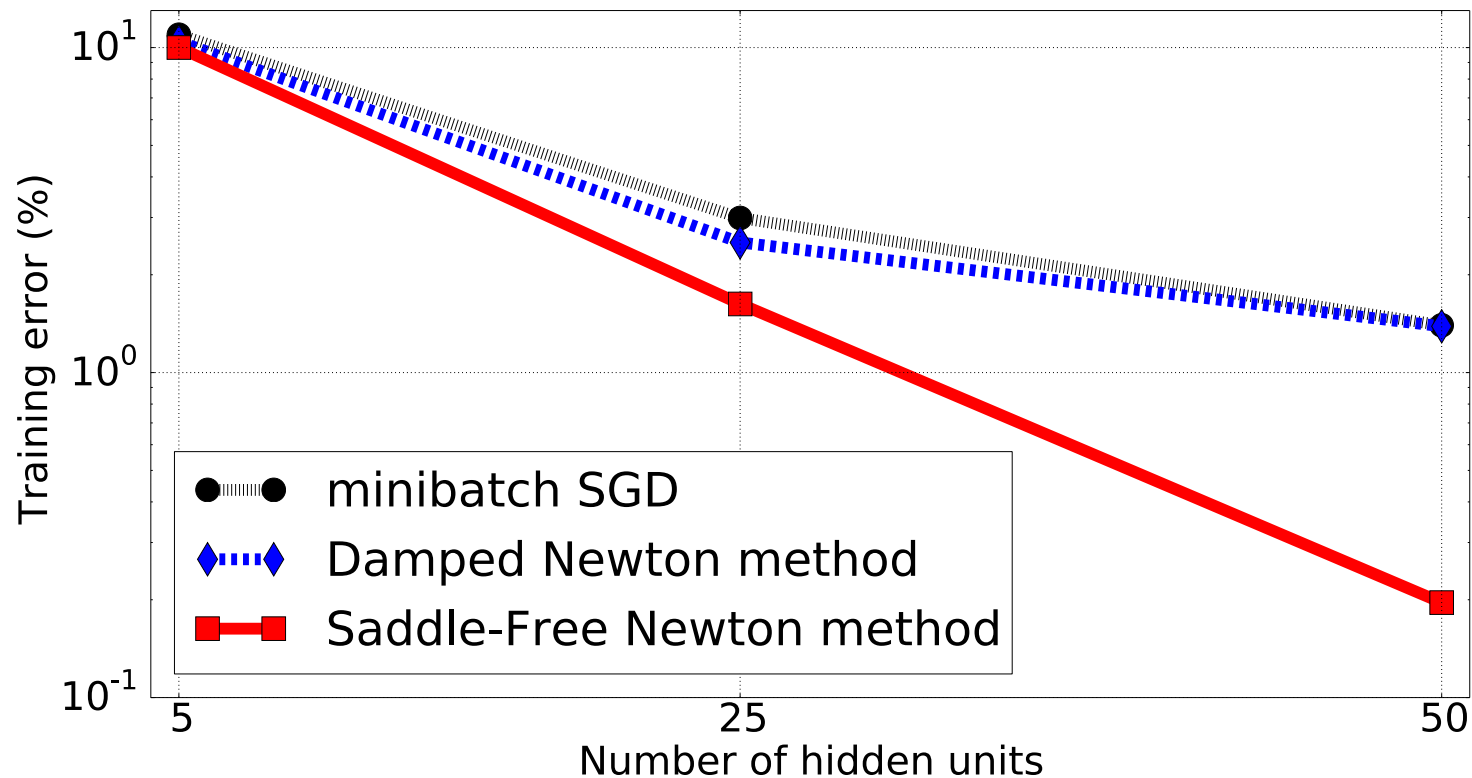
- Traditional thinking is that major obstacle for training deep nets is local minima
- Theoretical and empirical evidence suggest instead that saddle points are exponentially more prevalent critical points, and local minima tend to be of cost near that of global minimum
- (Pascanu, Dauphin, Ganguli, Bengio 2014): *On the saddle point problem for non-convex optimization.*



# Saddle-Free Optimization

(Pascanu, Dauphin, Ganguli, Bengio 2014)

- Replace eigenvalues  $\lambda$  of Hessian by  $|\lambda|$

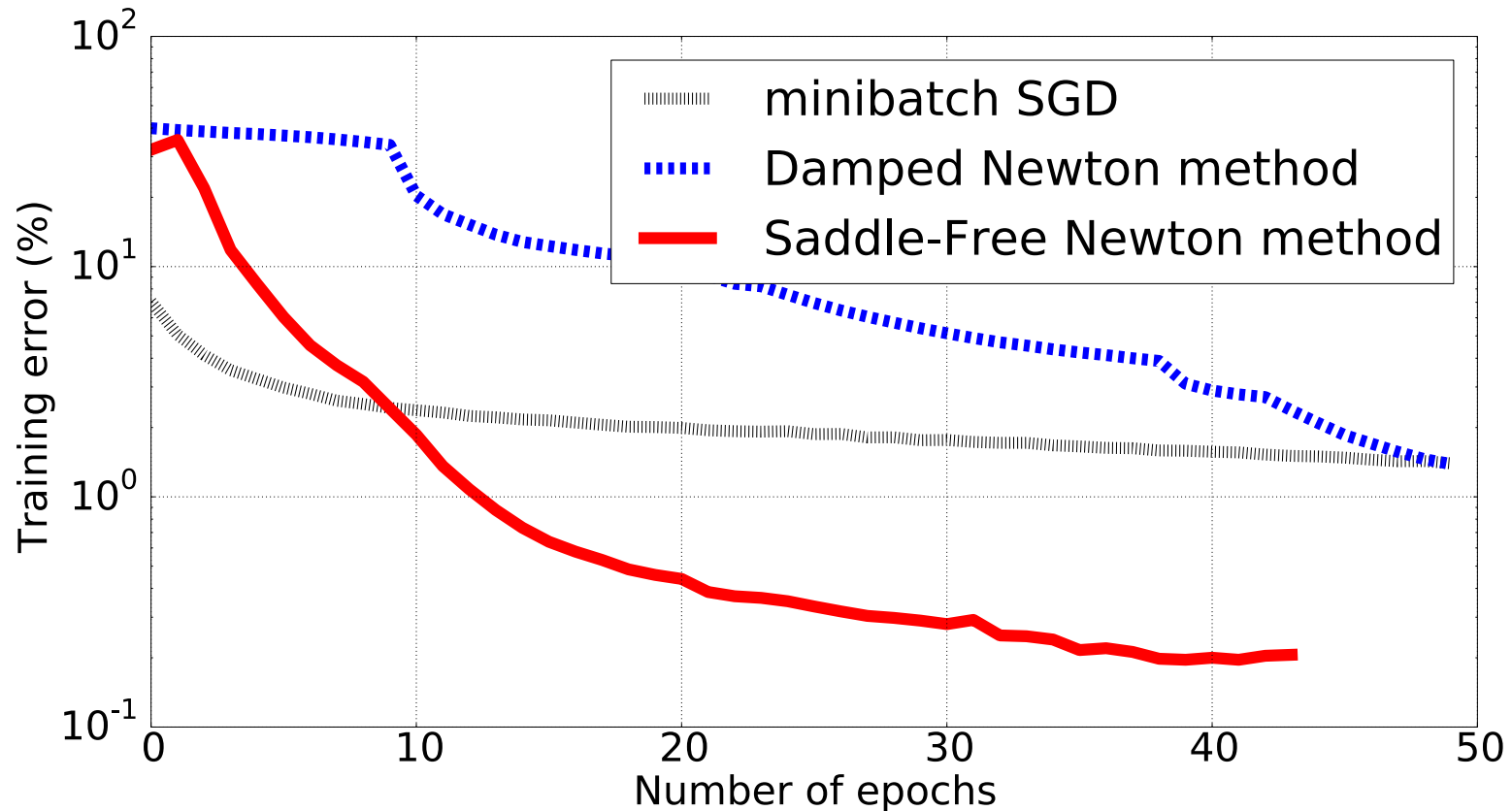




# Saddle-Free Optimization

(Pascanu, Dauphin, Ganguli, Bengio 2014)

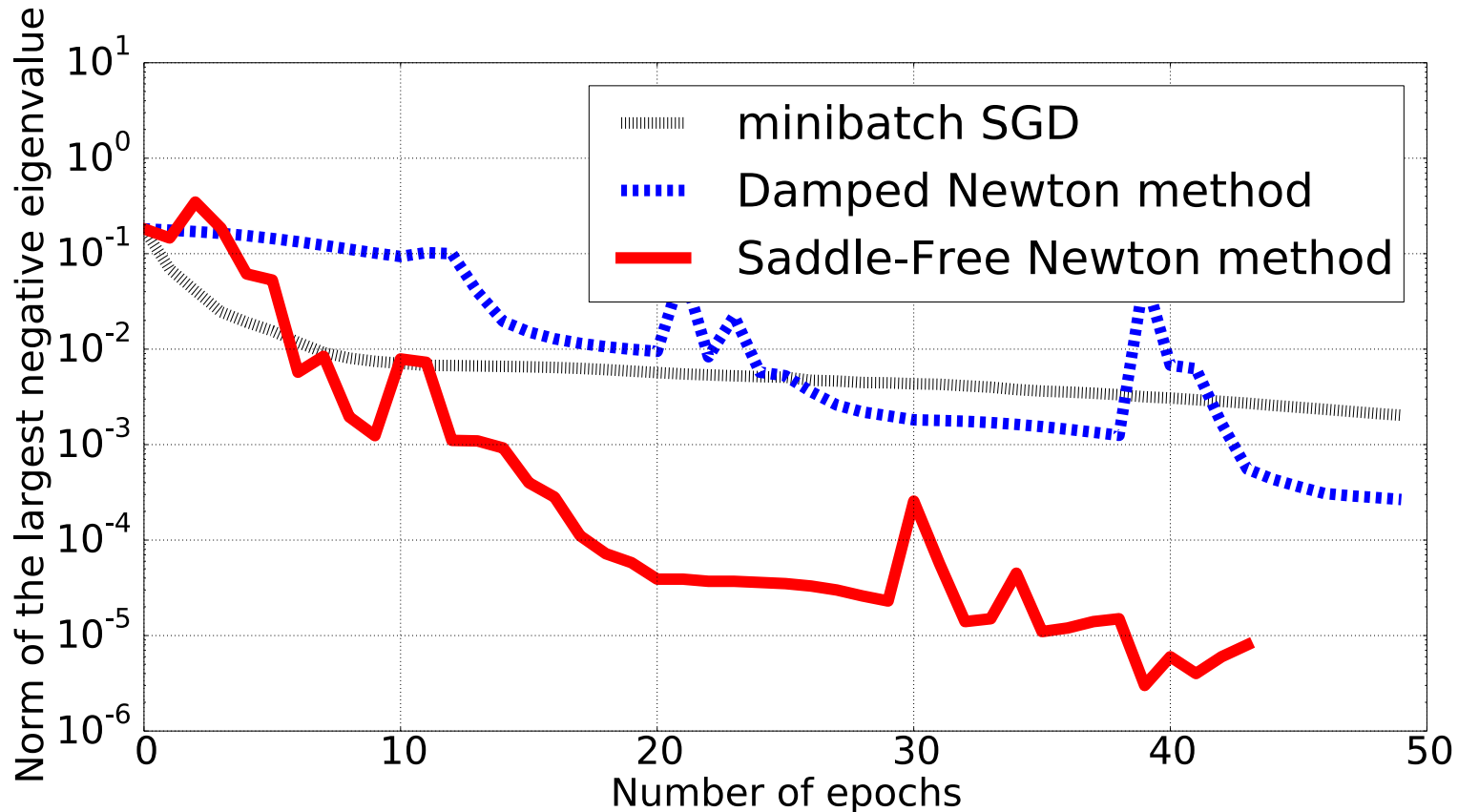
- Replace eigenvalues  $\lambda$  of Hessian by  $|\lambda|$



# Saddle-Free Optimization

(Pascanu, Dauphin, Ganguli, Bengio 2014)

- Replace eigenvalues  $\lambda$  of Hessian by  $|\lambda|$



# Guided Training, Intermediate Concepts

- In (Gulcehre & Bengio ICLR'2013) we set up a task that seems almost impossible to learn by shallow nets, deep nets, SVMs, trees, boosting etc

Algorithm	20k dataset		40k dataset		80k dataset	
	Training Error	Test Error	Training Error	Test Error	Training Error	Test Error
SVM RBF	26.2	50.2	28.2	50.2	30.2	49.6
KNN	24.7	50.0	25.3	49.5	25.6	49.0
Decision Tree	5.8	48.6	6.3	49.4	6.9	49.9
Randomized Trees	3.2	49.8	3.4	50.5	3.5	49.1
MLP	26.5	49.3	33.2	49.9	27.2	50.1
Convnet/Lenet5	50.6	49.8	49.4	49.8	50.2	49.8
Maxout Convnet	14.5	49.5	0.0	50.1	0.0	44.6
2 layer sDA	49.4	50.3	50.2	50.3	49.7	50.3
Struct. Supervised MLP w/o hints	0.0	48.6	0.0	36.0	0.0	12.4
Struct. MLP+CAE Supervised Finetuning	50.5	49.7	49.8	49.7	50.3	49.7
Struct. MLP+CAE+DAE Supervised Finetuning	49.1	49.7	49.4	49.7	50.1	49.7
Struct. MLP+DAE+DAE Supervised Finetuning	49.5	50.3	49.7	49.8	50.3	49.7



CHANCE  
PREDICTIONS

# Guided Training, Intermediate Concepts

- In (Gulcehre & Bengio ICLR'2013) we set up a task that seems almost impossible to learn by shallow nets, deep nets, SVMs, trees, boosting etc

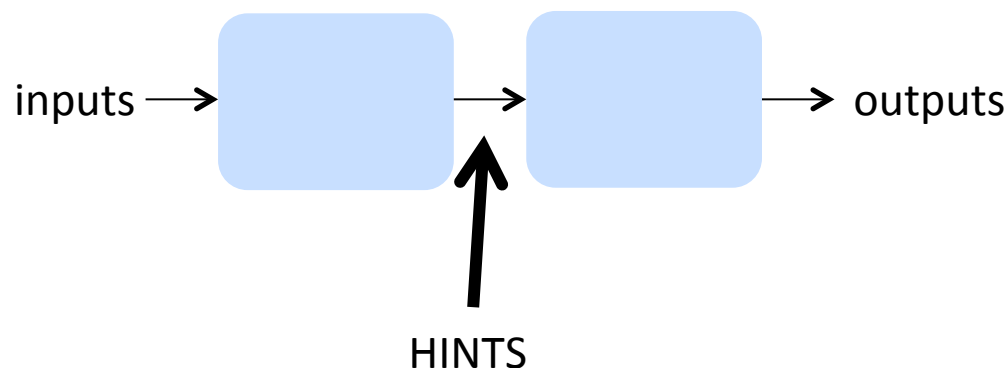
Algorithm	20k dataset		40k dataset		80k dataset	
	Training Error	Test Error	Training Error	Test Error	Training Error	Test Error
SVM RBF	26.2	50.2	28.2	50.2	30.2	49.6
KNN	24.7	50.0	25.3	49.5	25.6	49.0
Decision Tree	5.8	48.6	6.3	49.4	6.9	49.9
Randomized Trees	3.2	49.8	3.4	50.5	3.5	49.1
MLP	26.5	49.3	33.2	49.9	27.2	50.1
Convnet/Lenet5	50.6	49.8	49.4	49.8	50.2	49.8
Maxout Convnet	14.5	49.5	0.0	50.1	0.0	44.6
2 layer sDA	49.4	50.3	50.2	50.3	49.7	50.3
Struct. Supervised MLP w/o hints	0.0	48.6	0.0	36.0	0.0	12.4
Struct. MLP+CAE Supervised Finetuning	50.5	49.7	49.8	49.7	50.3	49.7
Struct. MLP+CAE+DAE Supervised Finetuning	49.1	49.7	49.4	49.7	50.1	49.7
Struct. MLP+DAE+DAE Supervised Finetuning	49.5	50.3	49.7	49.8	50.3	49.7
Struct. MLP with Hints	0.21	30.7	0	3.1	0	0.01

PERFECT  
PREDICTIONS



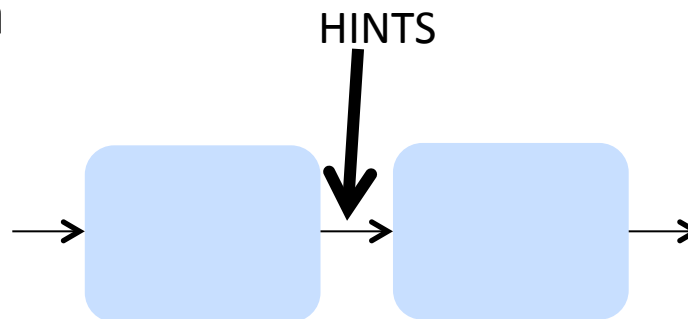
# Guided Training, Intermediate Concepts

- Breaking the problem in two sub-problems and pre-training each module separately, then fine-tuning, nails it
- *Need prior knowledge to decompose the task*
- **Guided pre-training** allows to find much better solutions, escape effective local minima



# Guided Training, Intermediate Concepts

- In (Gulcehre & Bengio ICLR'2013) we set up a task that seems almost impossible to learn by shallow nets, deep nets, SVMs, trees, boosting etc
- Breaking the problem in two sub-problems and pre-training each module separately, then fine-tuning, nails it
- *Need prior knowledge to decompose the task*
- **Guided pre-training** allows to find much better solutions, escape effective local minima



# Deep Learning Challenges

(Bengio, arxiv 1305.0445 Deep Learning of representations: Looking forward)

- Computational Scaling
- Optimization & Underfitting
- Intractable Marginalization, Approximate Inference & Sampling
- Disentangling Factors of Variation
- Reasoning & One-Shot Learning of Facts

# Why Unsupervised Learning?

- Recent progress mostly in supervised DL
- $\exists$  real challenges for unsupervised DL
- Potential benefits:
  - Exploit tons of unlabeled data
  - Answer new questions about the variables observed
  - Regularizer – transfer learning – domain adaptation
  - Easier optimization (local training signal)
  - Structured outputs



## Basic Challenge with Probabilistic Models: marginalization

- Joint and marginal likelihoods involve intractable sums over configurations of random variables (inputs  $x$ , latent  $h$ , outputs  $y$ ) e.g.

$$P(x) = \sum_h P(x,h)$$

$$P(x,h) = e^{-\text{energy}(x,h)} / Z$$

$$Z = \sum_{x,h} e^{-\text{energy}(x,h)}$$

- MCMC methods can be used for these sums, by sampling from a chain of  $x$ 's (or of  $(x,h)$  pairs) approximately from  $P(x,h)$

# MCMC Sampling Challenges

- Burn-in
  - Going from an unlikely configuration to likely ones



- **Mixing**

- Local: auto-correlation between successive samples



- Global: **mixing between major “modes”**

challenge



# Two Fundamental Problems with Probabilistic Models with Many Random Variables

1. MCMC mixing between modes (manifold hypothesis)



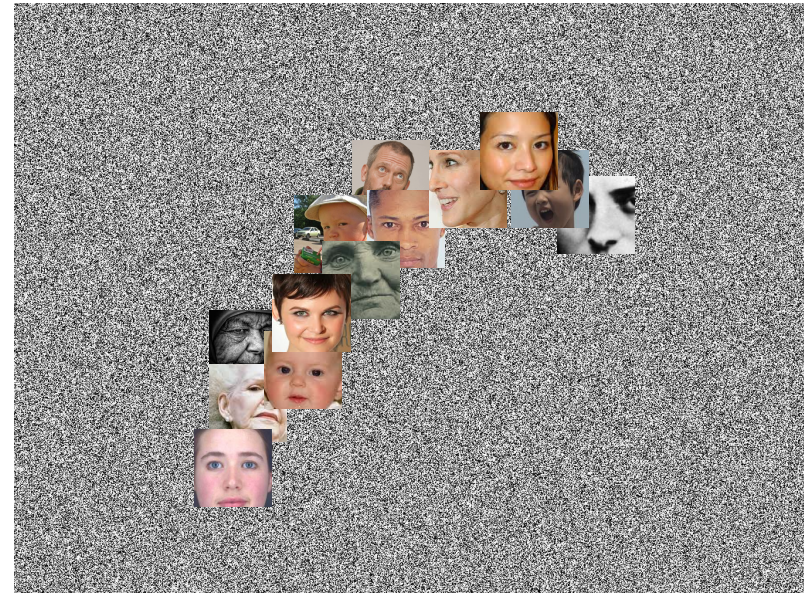
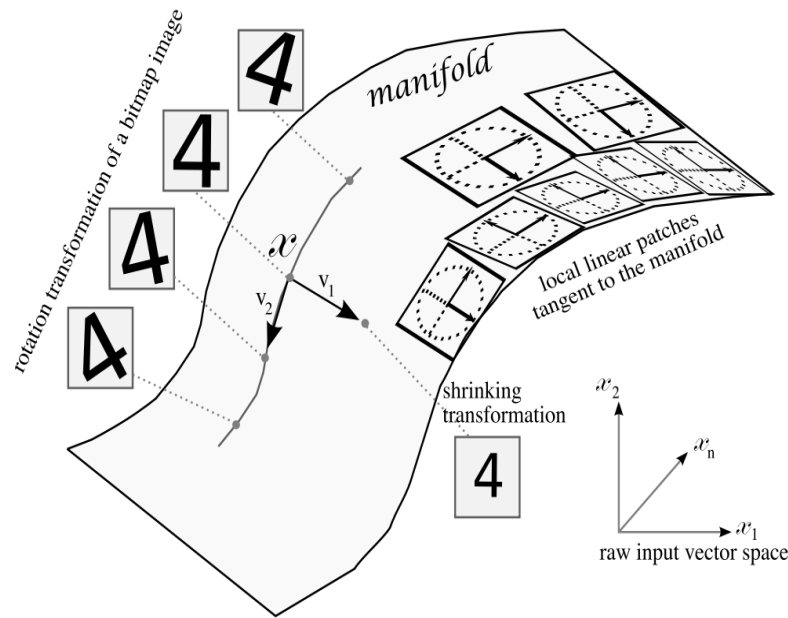
2. Many non-negligible modes (both in posterior & joint distributions)

# First Problem : MIXING BETWEEN SEPARATED MODES



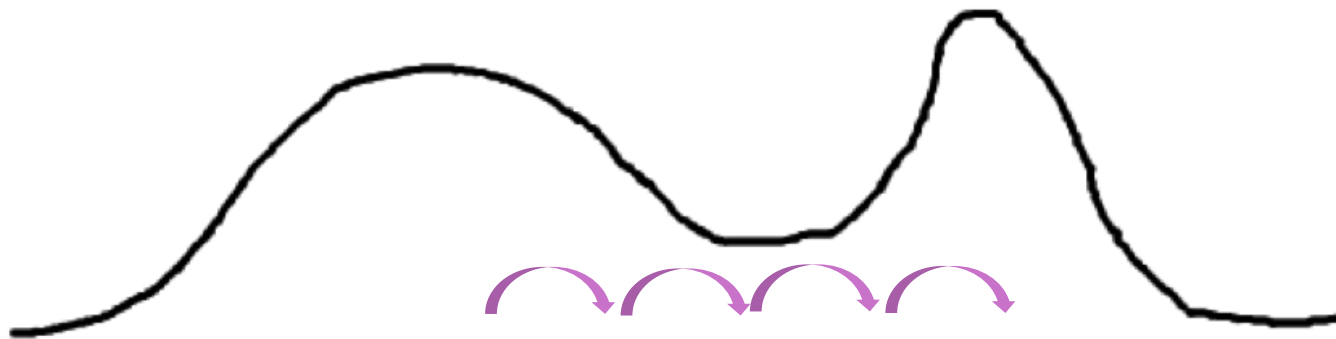
# Manifold = Mode

- 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)
- **Evidence: most input configurations are unlikely**

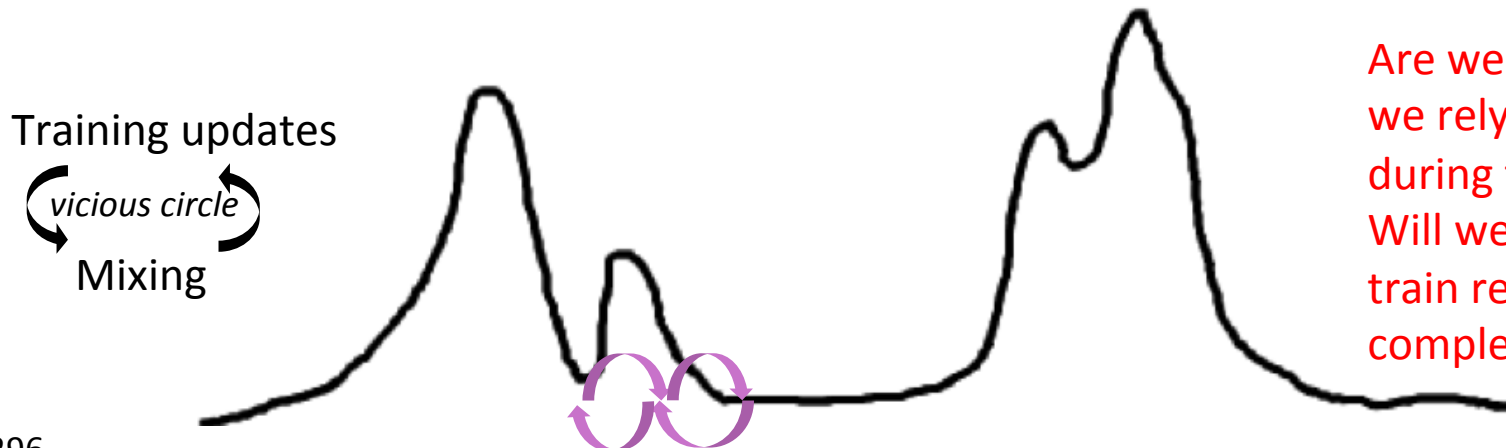


# For gradient & inference: More difficult to mix with better trained models

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



- Later on, hard to cross empty voids between modes



Are we doomed if  
we rely on MCMC  
during training?  
Will we be able to  
train really large &  
complex models?

# Fixing the Mixing Problem?

- If there were few important modes, we could just run many chains from different random starts and collect the results
- We have tried that and it helps only if there are few main modes
- Another option is **tempering** and related variants
- Appealing but very expensive, has not fixed the problem yet
- Deep representations: another promising avenue



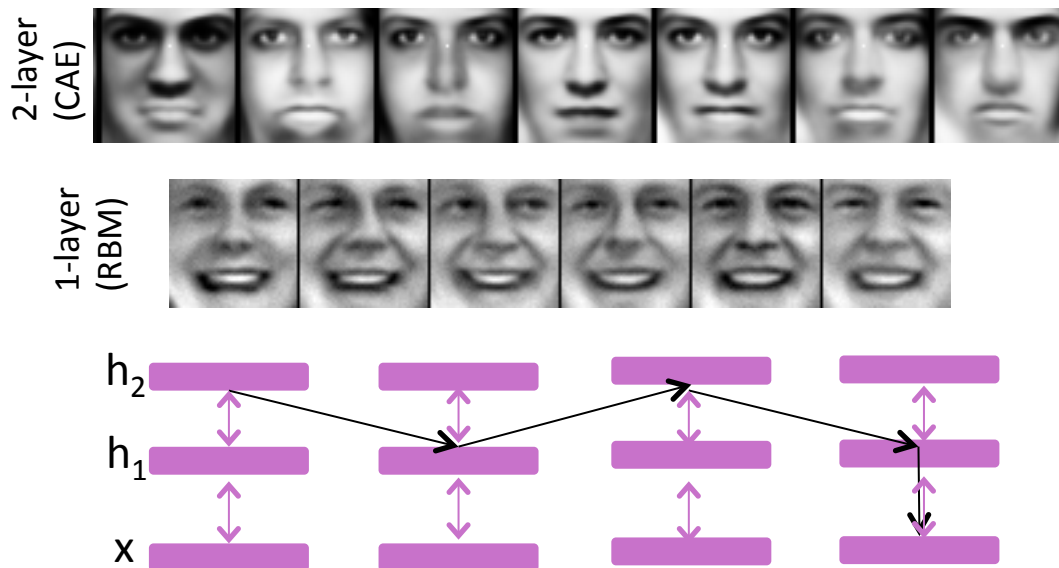
*How to temper chocolate*



# Poor Mixing: Depth to the Rescue

(Bengio et al ICML 2013)

- Sampling from DBNs and stacked Contractive Auto-Encoders:
  1. MCMC sampling from top layer model
  2. Propagate top-level representations to input-level repr.
- Deeper nets visit more modes (classes) faster



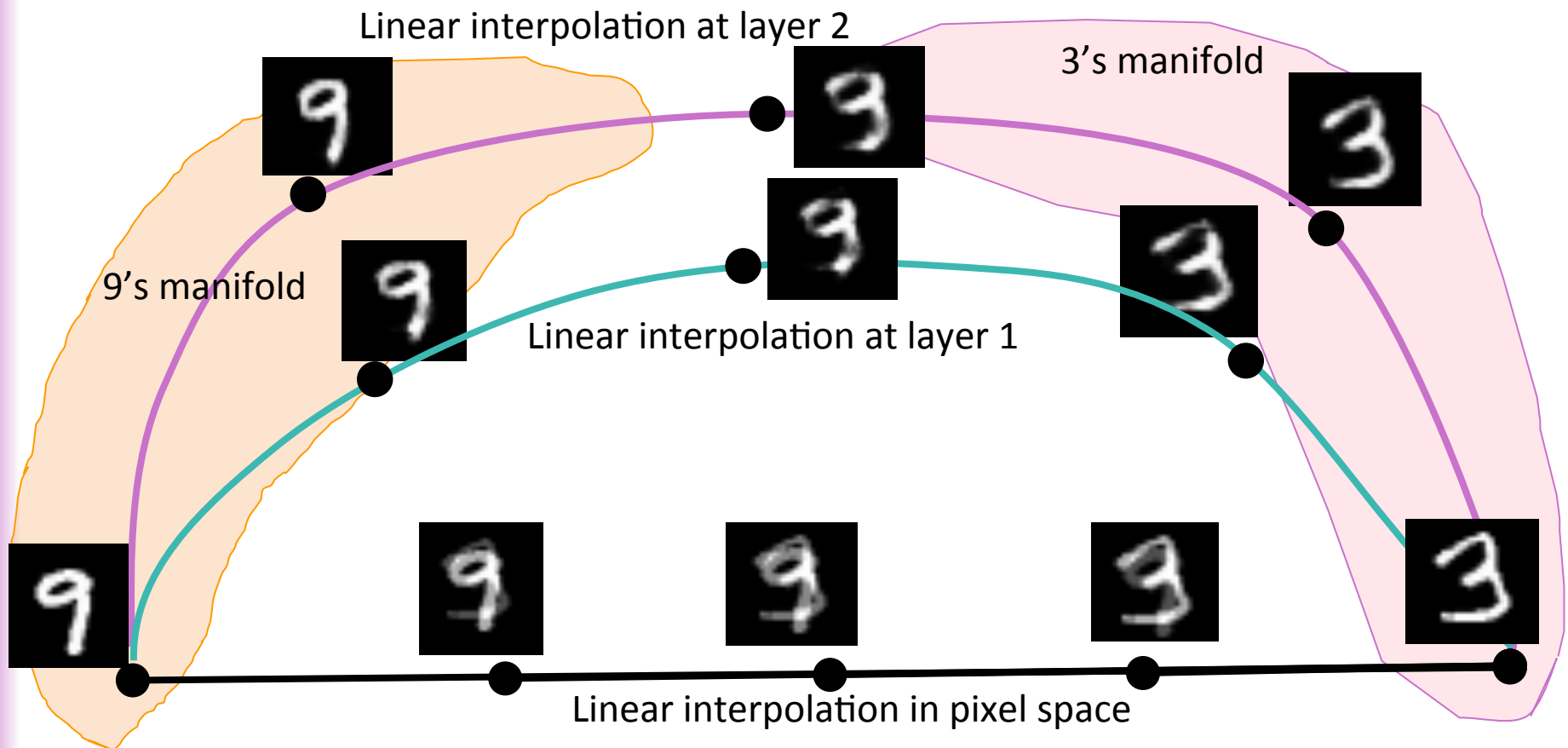


# Depth can help if MCMC is run in representation space

- Better mixing of MCMC at higher levels of representation (Bengio et al, ICML 2013)
- Distribution at higher levels of abstraction might be much simpler to represent (extreme: factorises)

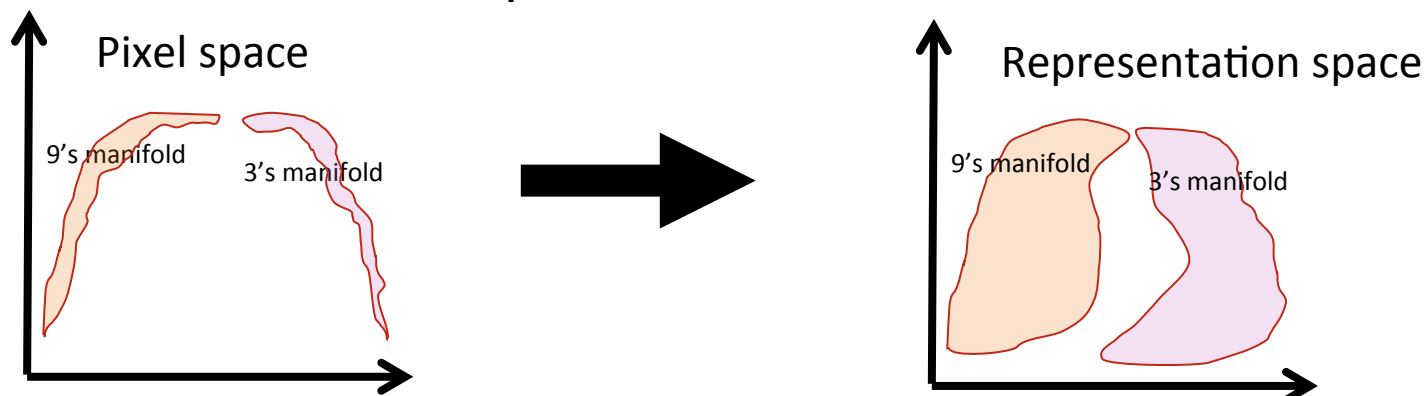
# Space-Filling in Representation-Space

- High-probability samples fill more the convex set between them when viewed in the learned representation-space, making the empirical distribution more uniform and unfolding manifolds



# Poor Mixing: Depth to the Rescue

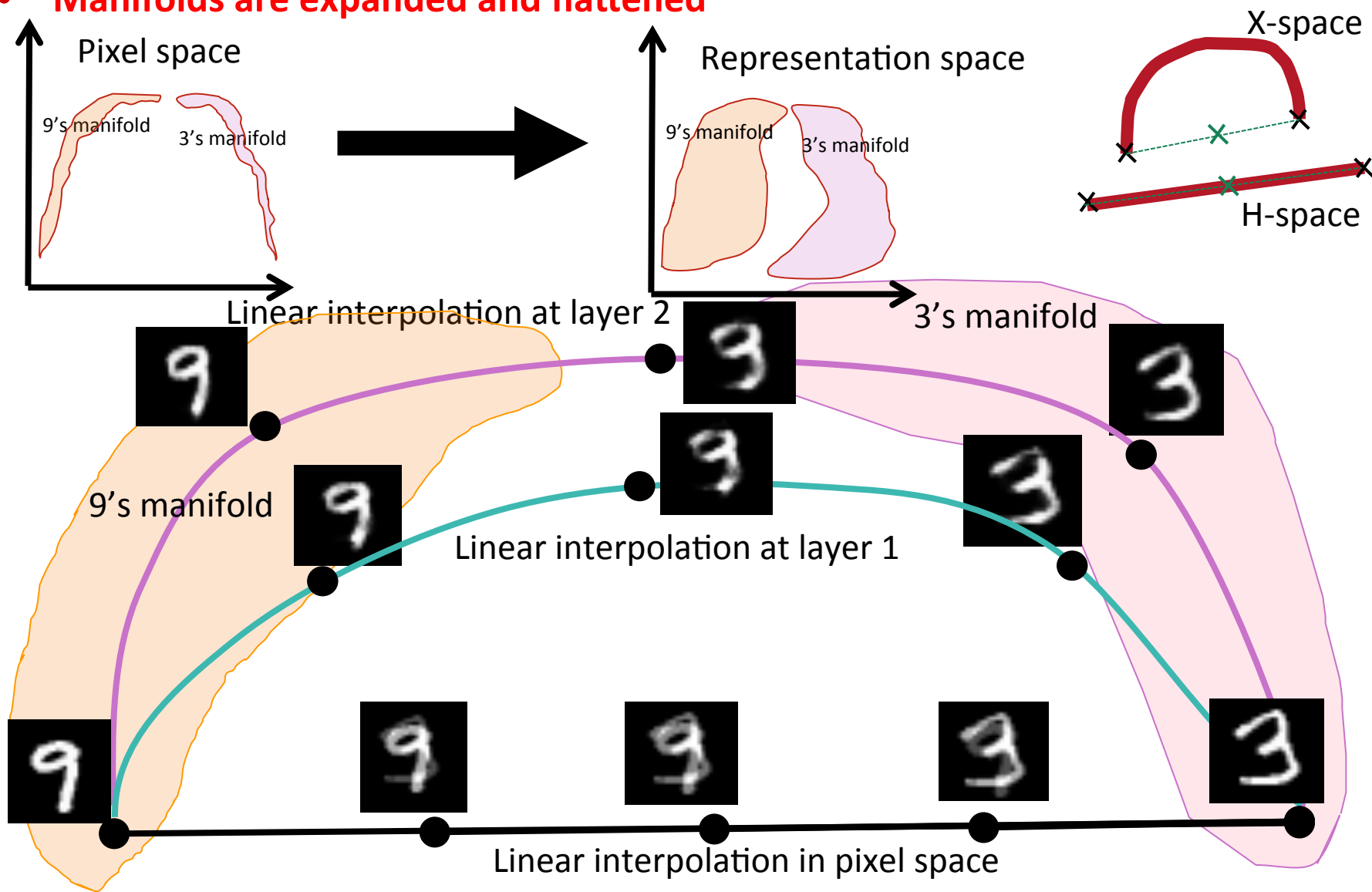
- Deeper representations  $\rightarrow$  abstractions  $\rightarrow$  disentangling
- E.g. reverse video bit, class bits in learned representations: easy to Gibbs sample between modes at abstract level
- Hypotheses tested and not rejected:
  - more abstract/disentangled representations unfold manifolds and fill more the space



- can be exploited for better mixing between modes

# Space-Filling in Representation-Space

- Deeper representations → abstractions → disentangling
- Manifolds are expanded and flattened



# Inference Challenges

- Many latent variables involved in understanding complex inputs (e.g. in NLP: sense ambiguity, parsing, semantic role)
- Almost any inference mechanism can be combined with deep learning
- See [Bottou, LeCun, Bengio 97], [Graves 2012]



- Complex inference can be hard (exponentially) and needs to be approximate → learn to perform inference

# The Main Problem that Remains: **MANY IMPORTANT MODES**



# Many Important Modes

- Issue arises typically in two places with probabilistic models:
  - **Inference**: need to consider the major modes of  $P(h|x)$  or  $P(y, h|x)$
  - **Learning** (estimating the log-likelihood gradient): need to consider the major modes of  $P(h, x)$  when computing the gradient of the normalization constant
- Important for:
  - *Unsupervised* (and semi-supervised) learning
  - *Structured output* learning

# Potentially **Huge** Number of Modes in the Posterior $P(h|x)$

- **Human** hears foreign speech  $x$ ,  $y$ =answer to question:
  - 10 word segments
  - 100 plausible candidates per word
  - $10^6$  possible segmentations
  - Most configurations (999999/1000000) implausible
  - ➔  $10^{20}$  high-probability modes
- Humans probably don't consider all these in their mind
- **All known approximate inference scheme break down if the posterior has a huge number of modes** (fails MAP & MCMC) and not respecting a variational approximation (fails variational)





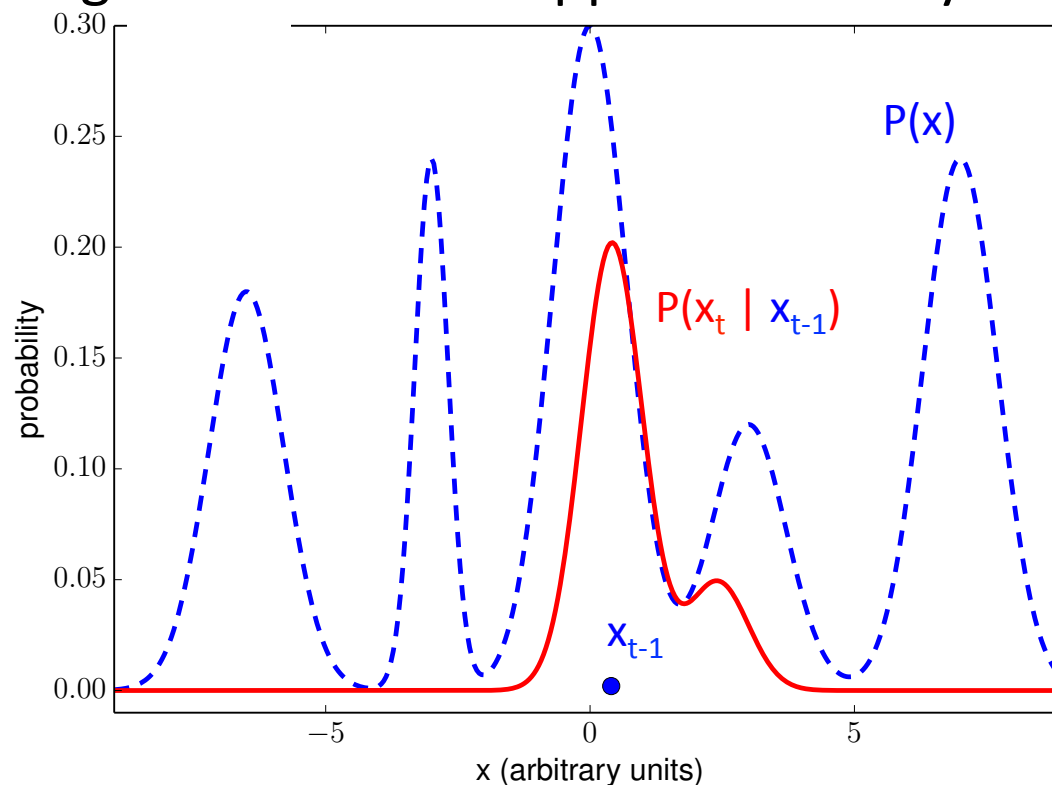
# PROPOSED SOLUTION

~~• Approximate inference?~~

• Function approximation

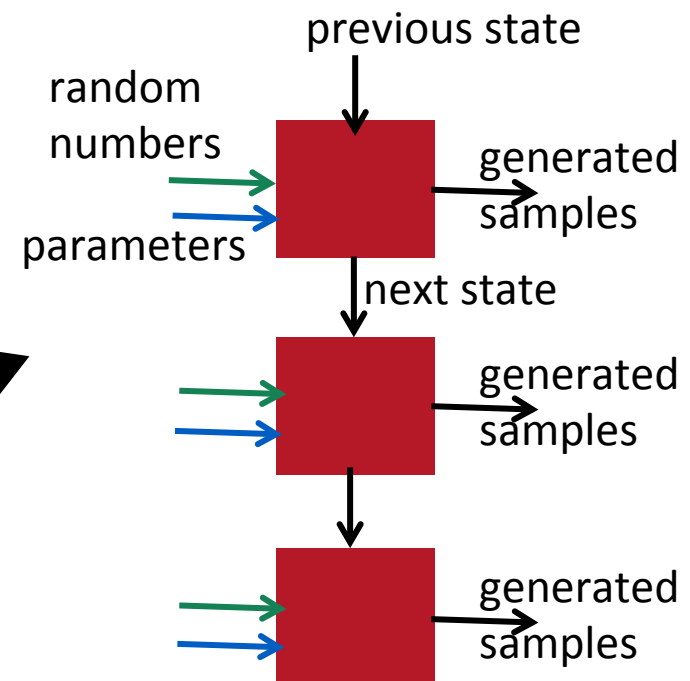
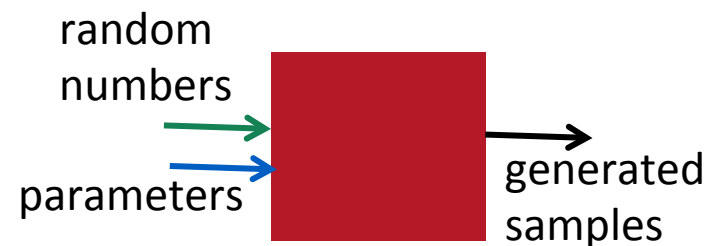
## Many Modes Challenge: Instead of Learning $P(x)$ directly, Learn Markov chain operator $P(x_t | x_{t-1})$

- $P(x)$  may have many modes, making the normalization constant intractable, and MCMC approximations poor
- Partition fn of  $P(x_t | x_{t-1})$  much simpler because most of the time a local move, might even be well approximated by unimodal



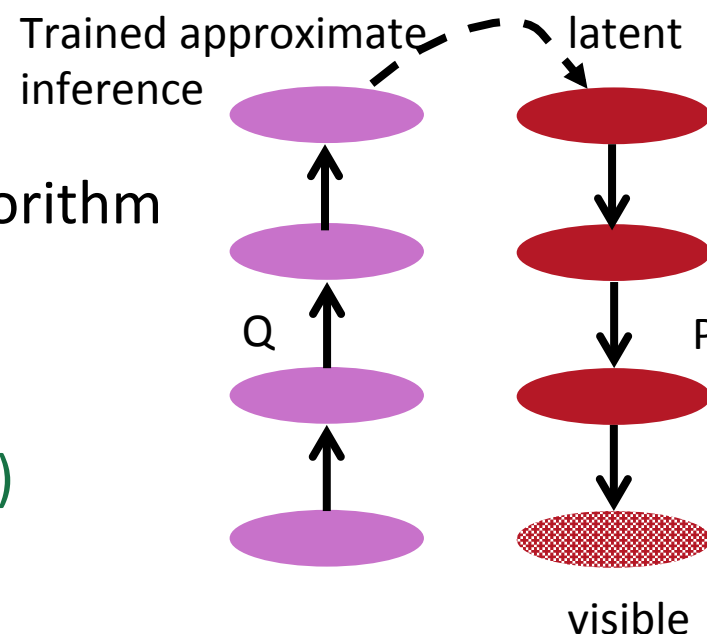
# Bypassing Normalization Constants with Generative Black Boxes

- **Instead of parametrizing  $p(x)$ , parametrize a machine which generates samples**
- (Goodfellow et al, 2014, Generative adversarial nets) for the case of ancestral sampling in a deep generative net
- (Bengio et al, ICML 2014, Generative Stochastic Networks), learning the transition operator of a Markov chain that generates the data



# Ancestral Sampling with Learned Approximate Inference

- Helmholtz machine & Wake-Sleep algorithm
  - (Dayan, Hinton, Neal, Zemel 1995)
- Variational Auto-Encoders
  - (Kingma & Welling 2013, ICLR 2014)
  - (Gregor et al ICML 2014)
  - (Rezende et al ICML 2014)
  - (Mnih & Gregor ICML 2014)
- Reweighted Wake-Sleep
  - (Bornschein & Bengio 2014)



# Deep Learning Challenges

(Bengio, arxiv 1305.0445 Deep Learning of representations: Looking forward)

- Computational Scaling
- Optimization & Underfitting
- Intractable Marginalization, Approximate Inference & Sampling
- Disentangling Factors of Variation
- Reasoning & One-Shot Learning of Facts

# Disentangling the Underlying Factors

- How could a learner disentangle the unknown underlying factors of variation?
  - Statistical structure present in the data
  - Hints = priors
- Good disentangling →  
avoid the curse of dimensionality

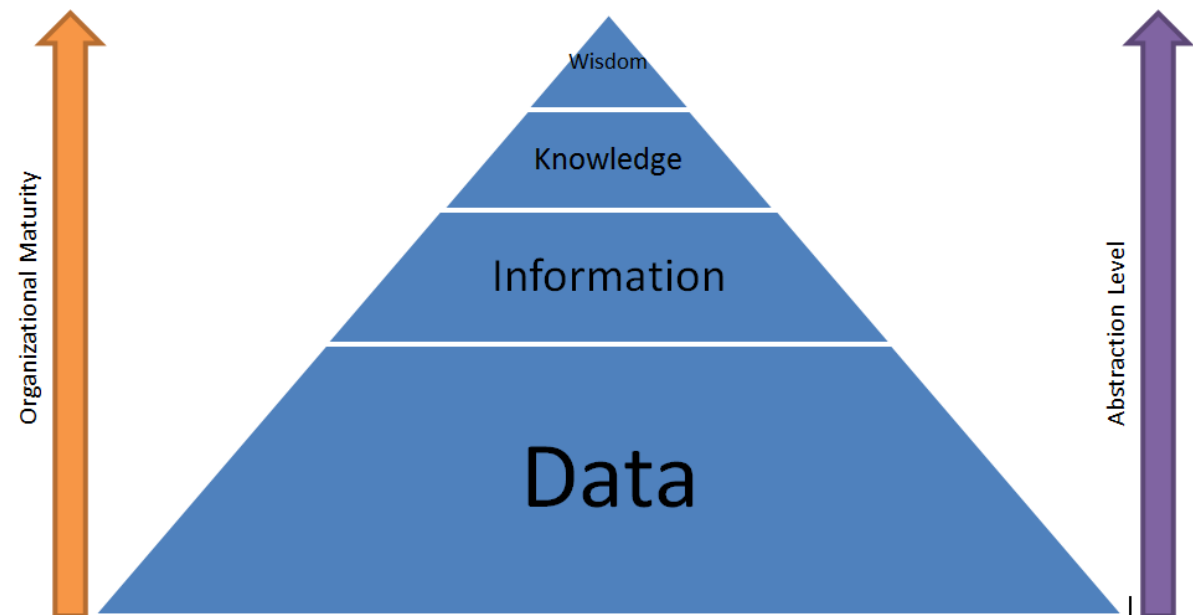


# Broad Priors as Hints to Disentangle the Factors of Variation

- *Multiple factors*: distributed representations
- Multiple levels of abstraction: *depth*
- *Semi-supervised* learning: Y is one of the factors explaining X
- *Multi-task* learning: different tasks share some factors
- *Manifold* hypothesis: probability mass concentration
- Natural *clustering*: class = manifold, well-separated manifolds
- Temporal and spatial *coherence*
- *Sparsity*: most factors irrelevant for particular X
- *Simplicity* of factor dependencies (in the right representation)

# Learning Multiple Levels of Abstraction

- The big payoff of deep learning is to allow learning higher levels of abstraction
- Higher-level abstractions disentangle the factors of variation, which allows much easier generalization and transfer





# Conclusions

- Deep Learning has matured
  - Int. Conf. on Learning Representation 2013 a huge success!
- Industrial applications (Google, Microsoft, Baidu, Facebook, ...)
- Room for improvement:
  - Scaling computation
  - Optimization
  - Bypass intractable marginalizations
  - More disentangled abstractions
  - Reason from incrementally added facts

If Time Permits...

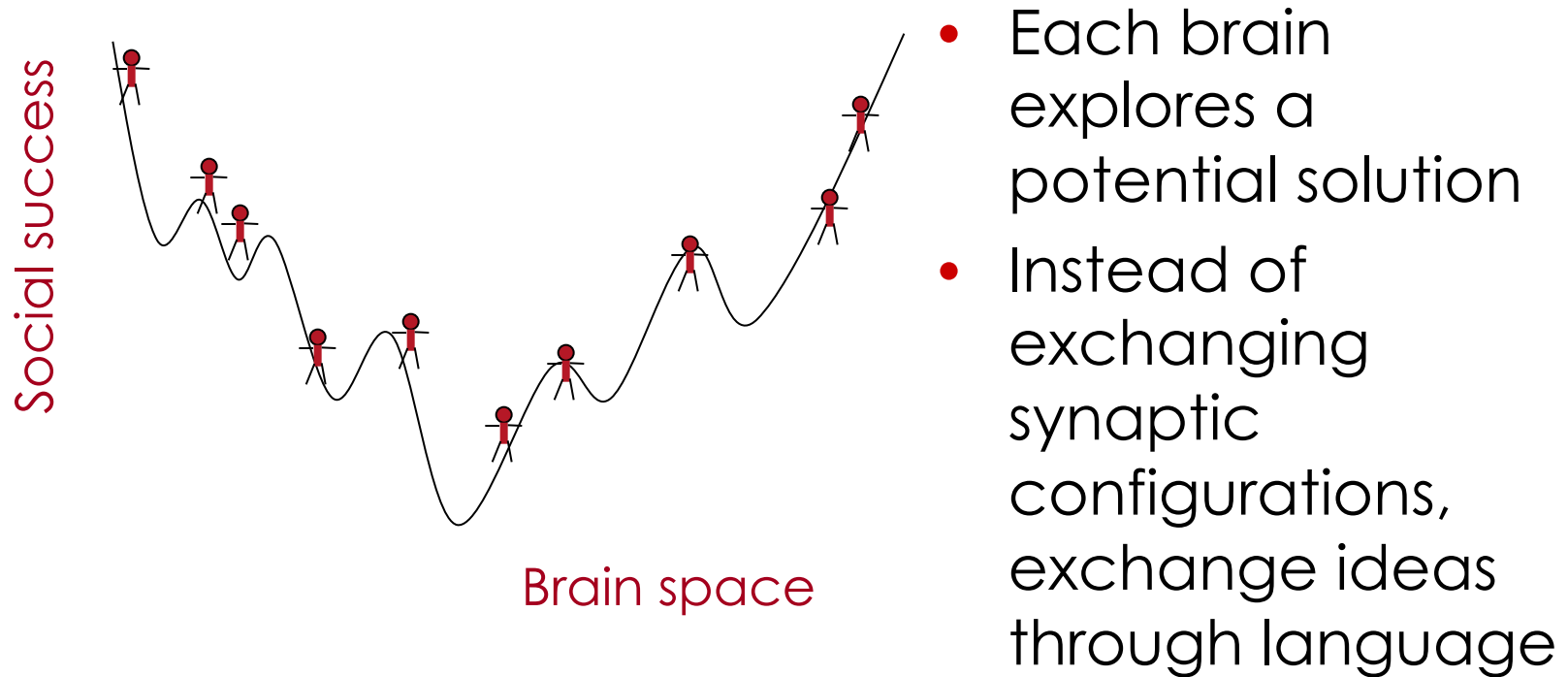
Issue: **underfitting** due to combinatorially many poor *effective* local minima

↖  
where the optimizer gets stuck

## Culture vs Effective Local Minima

Bengio 2013 (also arXiv 2012)

# Parallelized exploration in brain space



# Memes

Genetic Algorithms

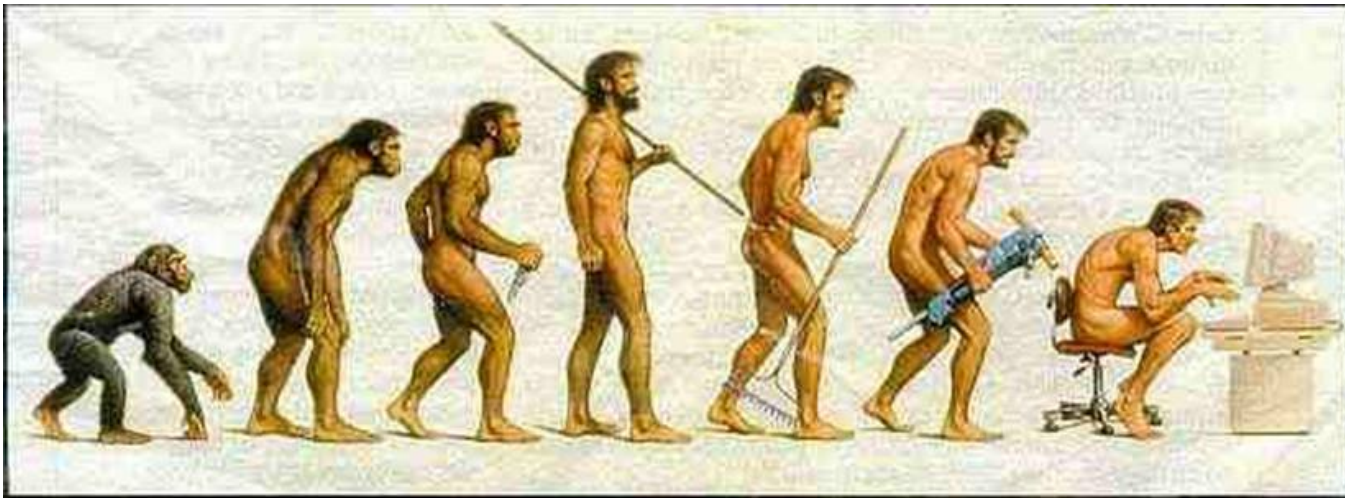
Population of candidate solutions

Recombination mechanism

Evolution of ideas

Brains

Culture and language



## Hypothesis 1

- When the brain of a single biological agent learns, it performs an approximate optimization with respect to some endogenous objective.

## Hypothesis 2

- When the brain of a single biological agent learns, it relies on approximate local descent in order to gradually improve itself.

Theoretical and experimental results on deep learning suggest:

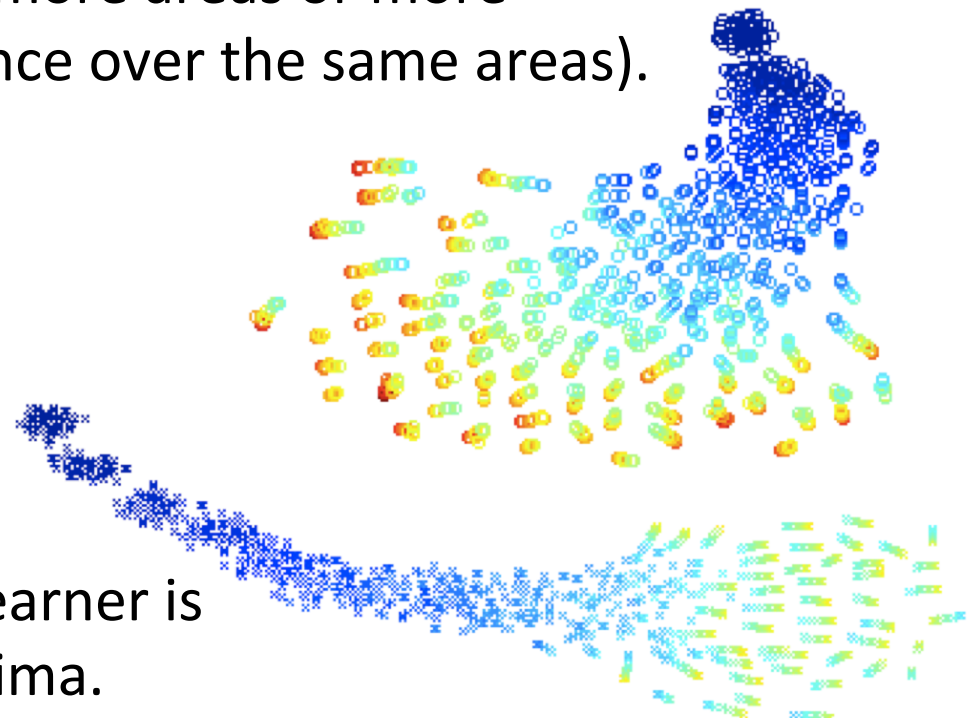
## Hypothesis 3

- Higher-level abstractions in brains are represented by deeper computations (going through more areas or more computational steps in sequence over the same areas).

## Hypothesis 4

- Learning of a single human learner is limited by *effective* local minima.

Possibly due to ill-conditioning, but behaves like local min



## Hypothesis 5

- A single human learner is unlikely to discover high-level abstractions by chance because these are represented by a deep sub-network in the brain.

## Hypothesis 6

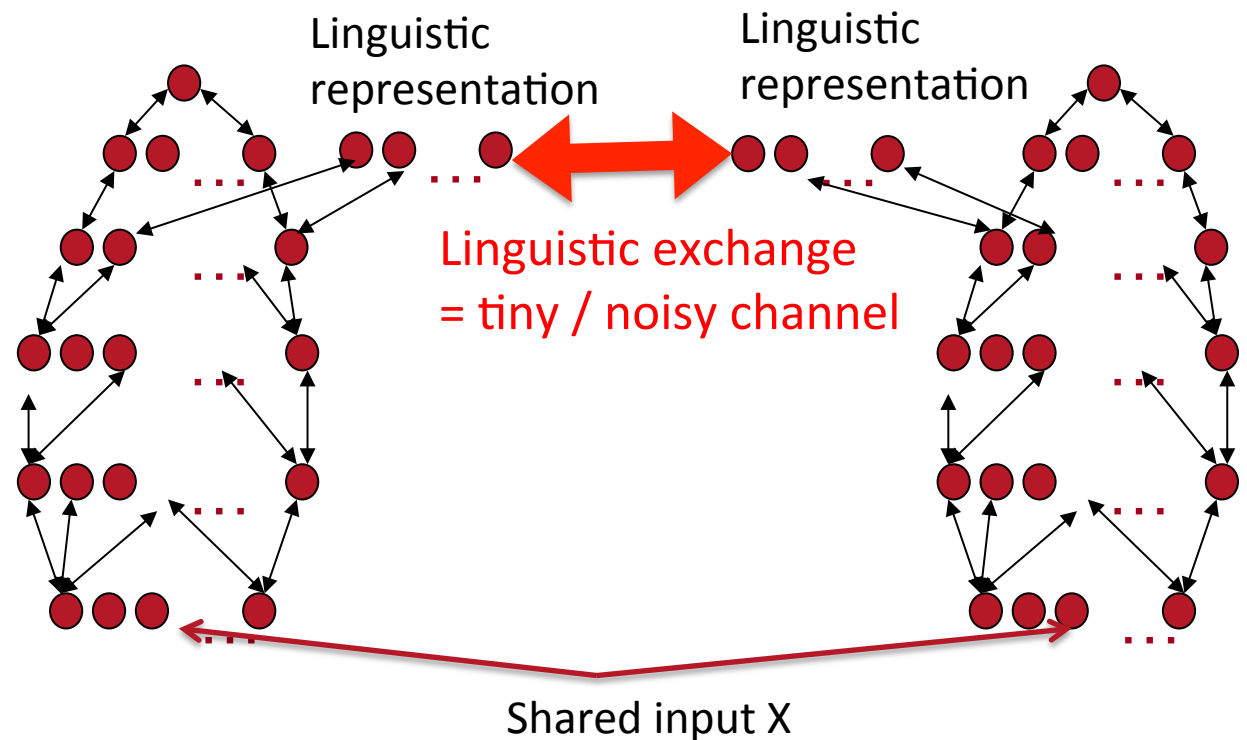
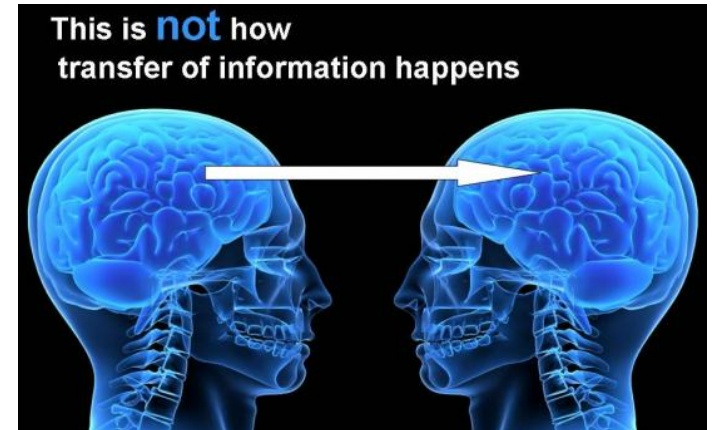
Curriculum learning  
(Bengio et al ICML 2009)

- A human brain can learn high-level abstractions if guided by the signals produced by other humans, which act as hints or indirect supervision for these high-level abstractions.

Supporting evidence: (Gulcehre & Bengio ICLR 2013)



# How is one brain transferring abstractions to another brain?



# How do we escape local minima?

- linguistic inputs = extra examples, summarize knowledge
- criterion landscape easier to optimize (e.g. curriculum learning)
- turn difficult unsupervised learning into easy supervised learning of intermediate abstractions

How could language/education/culture possibly help find the better local minima associated with more useful abstractions?

## Hypothesis 7

More than random search:  
potential exponential speed-up by divide-and-conquer  
combinatorial advantage:  
can combine solutions to  
independently solved sub-problems

- Language and meme recombination provide an efficient **evolutionary operator**, allowing rapid search in the space of **memes**, that helps humans build up better high-level internal representations of their world.

# From where do new ideas emerge?

- Seconds: **inference** (novel explanations for current  $x$ )
- Minutes, hours: **learning** (local descent, like current DL)
- Years, centuries: **cultural evolution** (global optimization, recombination of ideas from other humans)

# Related Tutorials

- Deep Learning tutorials (python): <http://deeplearning.net/tutorials>
- Stanford deep learning tutorials with simple programming assignments and reading list  
<http://deeplearning.stanford.edu/wiki/>
- ACL 2012 Deep Learning for NLP tutorial  
<http://www.socher.org/index.php/DeepLearningTutorial/>
- ICML 2012 Representation Learning tutorial  
<http://www.iro.umontreal.ca/~bengioy/talks/deep-learning-tutorial-2012.html>
- IPAM 2012 Summer school on Deep Learning  
<http://www.iro.umontreal.ca/~bengioy/talks/deep-learning-tutorial-aaai2013.html>
- **More reading: Paper references in separate pdf, on my web page**

# Software

- Theano (Python CPU/GPU) mathematical and deep learning library <http://deeplearning.net/software/theano>
  - Can do automatic, symbolic differentiation
- Senna: POS, Chunking, NER, SRL
  - by Collobert et al. <http://ronan.collobert.com/senna/>
  - State-of-the-art performance on many tasks
  - 3500 lines of C, extremely fast and using very little memory
- Torch ML Library (C++ + Lua) <http://www.torch.ch/>
- Recurrent Neural Network Language Model  
<http://www.fit.vutbr.cz/~imikolov/rnnlm/>
- Recursive Neural Net and RAE models for paraphrase detection, sentiment analysis, relation classification [www.socher.org](http://www.socher.org)

## Software: what's next

- Off-the-shelf SVM packages are useful to researchers from a wide variety of fields (no need to understand RKHS).
- To make deep learning more accessible: release off-the-shelf learning packages that handle hyper-parameter optimization, exploiting multi-core or cluster at disposal of user.
  - Spearmint (Snoek)
  - HyperOpt (Bergstra)



LISA team: **Merci! Questions?**

