
Representation Learning and Deep Learning

Yoshua Bengio

Institute for Pure and Applied Mathematics (IPAM)
Graduate Summer School 2012 on deep learning
and feature learning

July 2012, UCLA

Université 
de Montréal



Outline of the Tutorial

1. Motivations and Scope
2. Algorithms
3. Analysis, Issues and Practice
4. Applications to NLP
5. Culture vs Local Minima

See (Bengio, Courville & Vincent 2012)

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

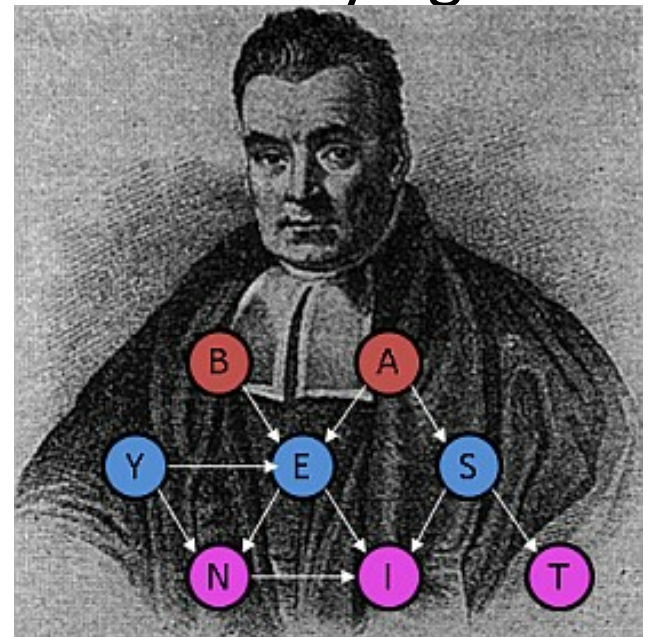
And <http://www.iro.umontreal.ca/~bengioy/talks/deep-learning-gss2012.html> for a pdf of the slides and a detailed list of references.

Ultimate Goals

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

Representation Learning

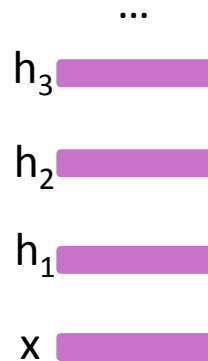
- Good features essential for successful ML
- Handcrafting features vs learning them
- Good representation: captures posterior belief about explanatory causes, disentangles these underlying factors of variation
- Representation learning: **guesses** the features / factors / causes = good representation.



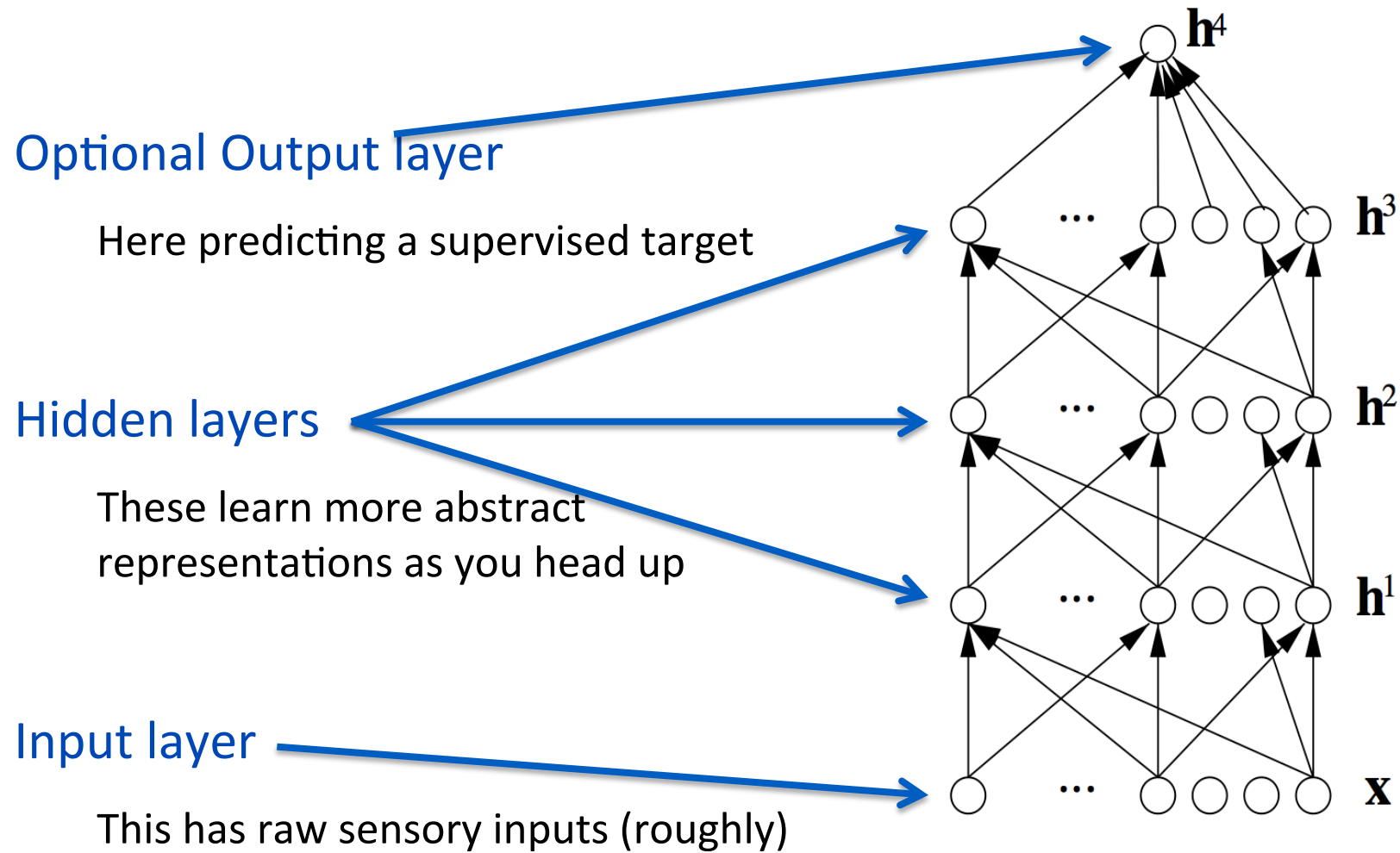
Deep Representation Learning

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

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



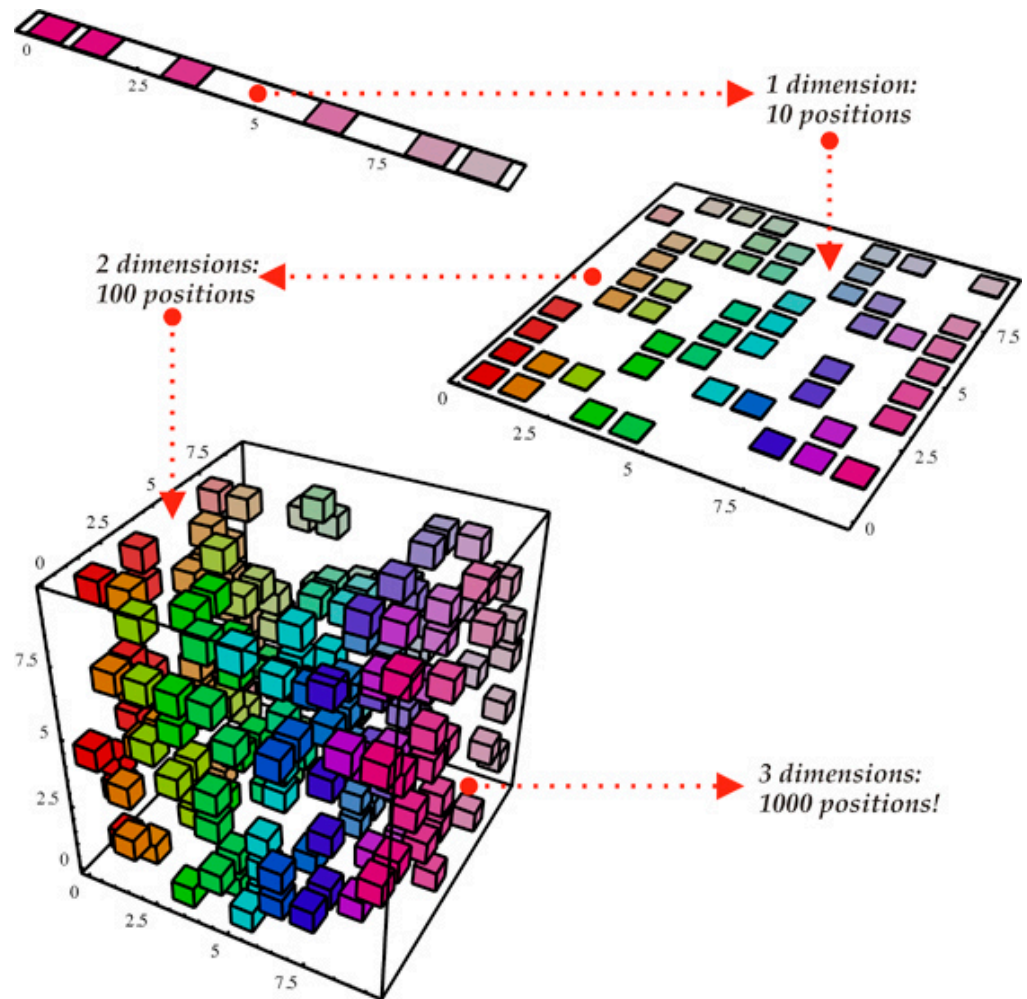
A Good Old Deep Architecture



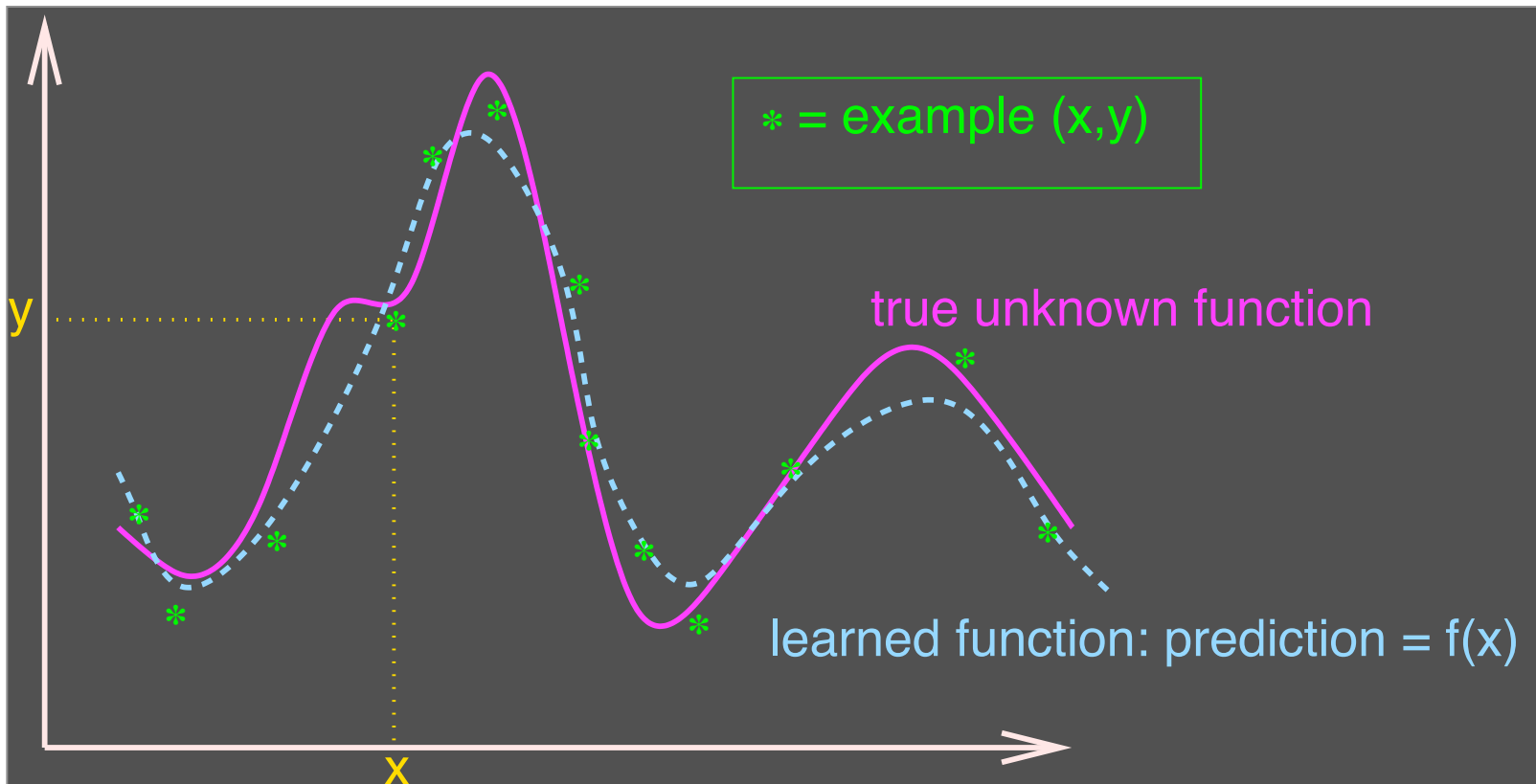
What We Are Fighting Against: The Curse of Dimensionality

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

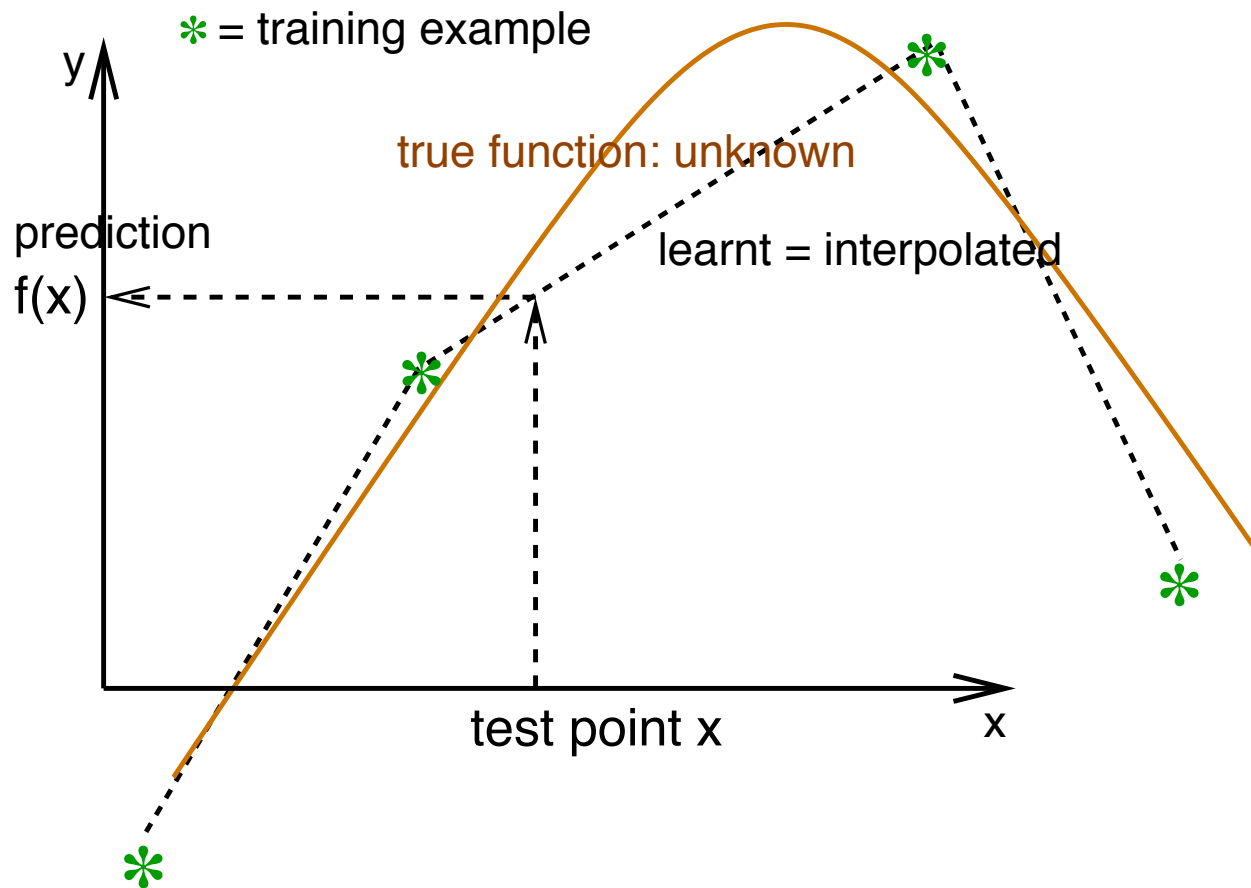
Classical solution: hope
for a smooth enough
target function, or
make it smooth by
handcrafting features



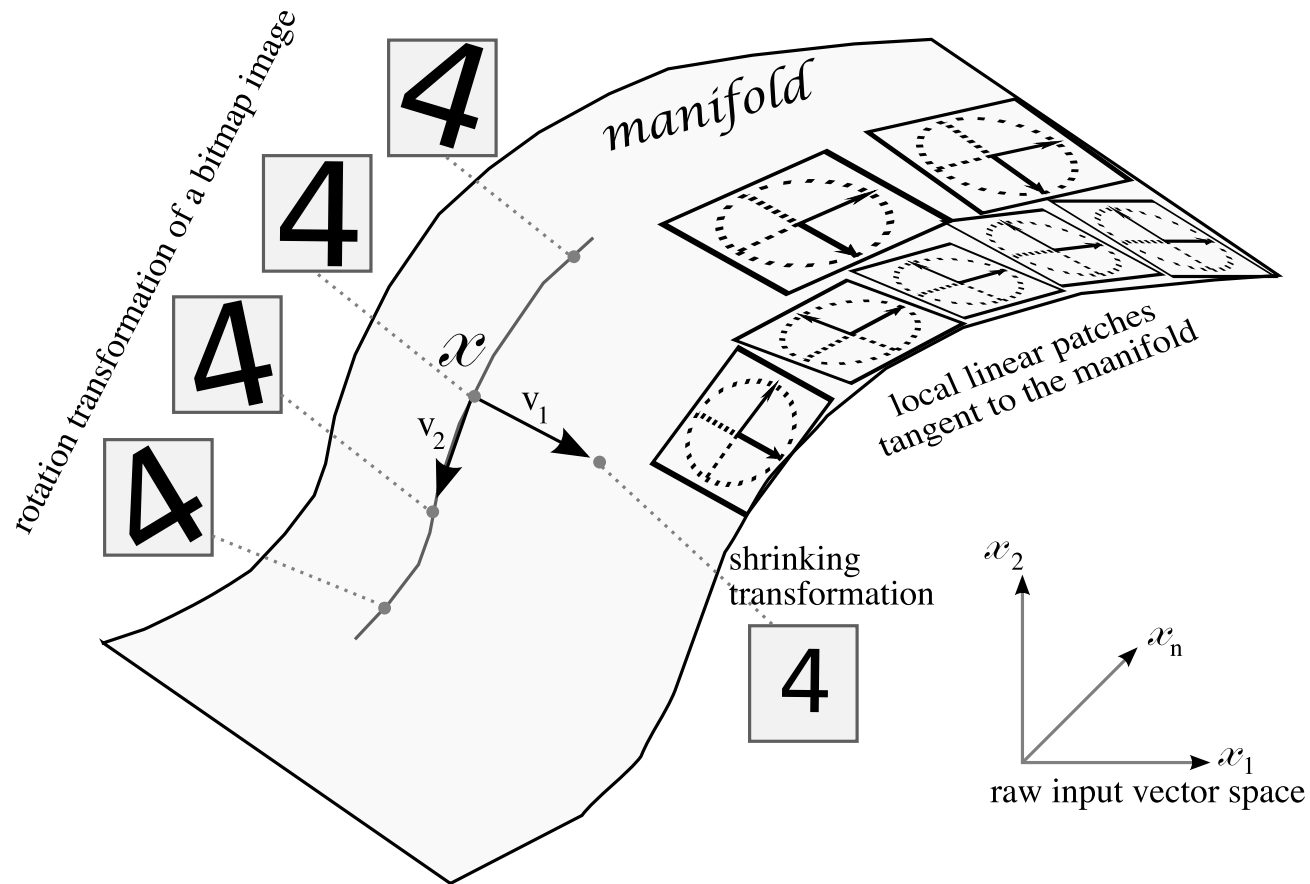
Easy Learning



Local Smoothness Prior: Locally Capture the Variations



Real Data Are on Highly Curved Manifolds

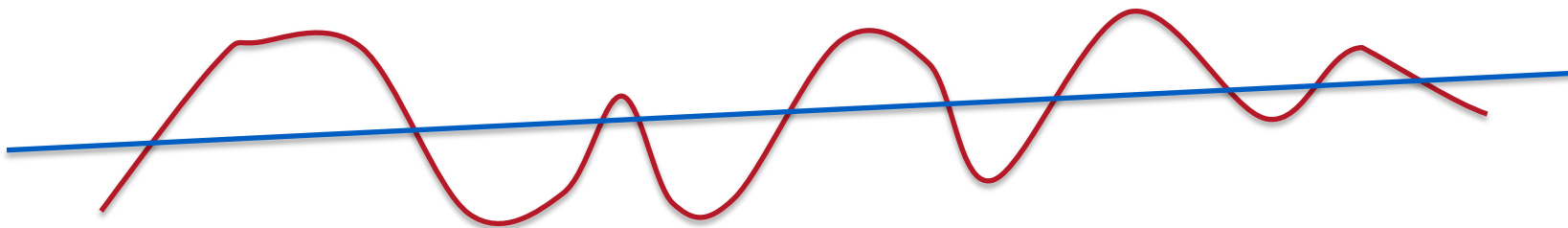


Not Dimensionality so much as Number of Variations



(Bengio, Delalleau & Le Roux 2007)

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



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

Is there any hope to
generalize non-locally?

Yes! Need more priors!

Part 1

Six Good Reasons to Explore Representation Learning

#1 Learning features, not just handcrafting them

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

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

“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

How can we automatically learn good features?

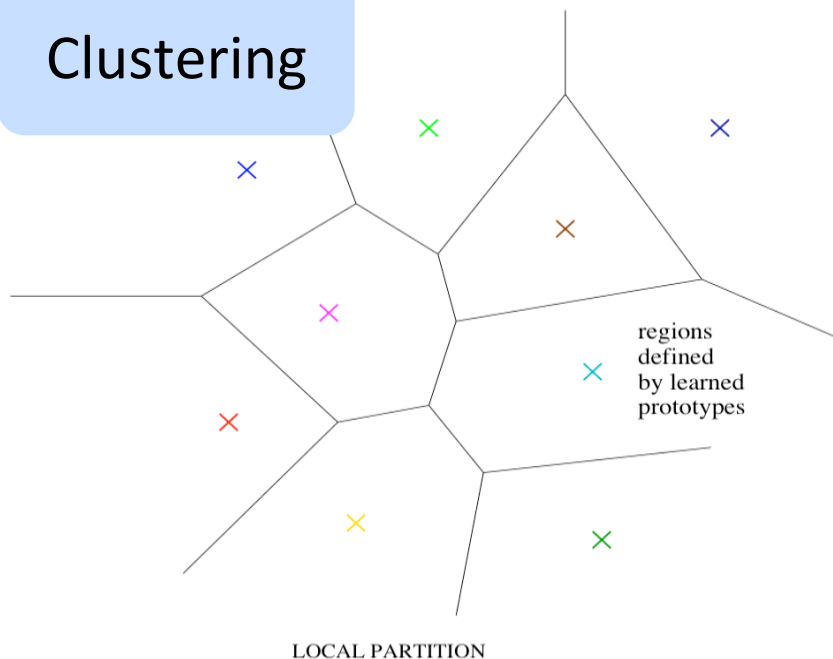
Claim: to approach AI, need to move scope of ML beyond hand-crafted features and simple models

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

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

#2 The need for distributed representations

Clustering

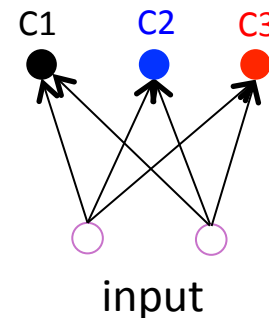
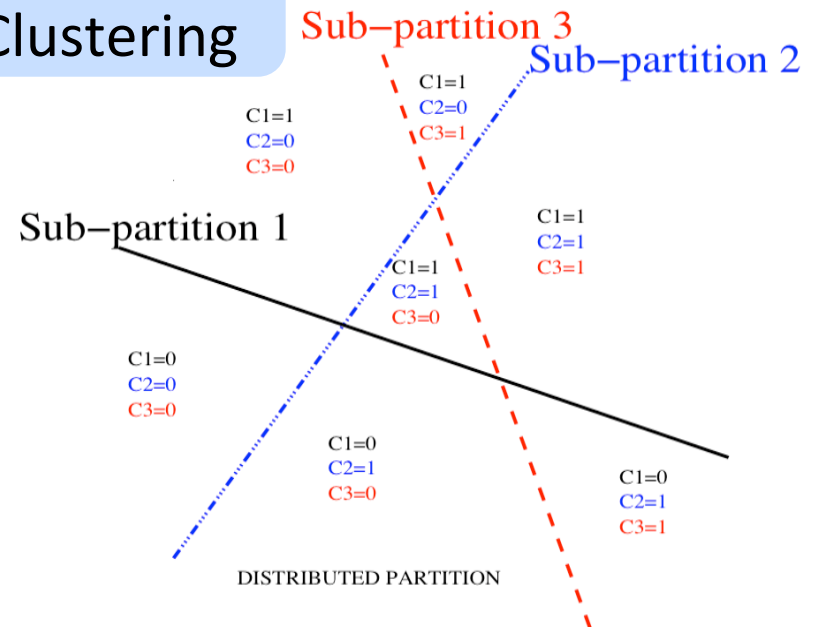


- Clustering, Nearest-Neighbors, RBF SVMs, local non-parametric density estimation & prediction, decision trees, etc.
- Parameters for each distinguishable region
- # distinguishable regions linear in # parameters

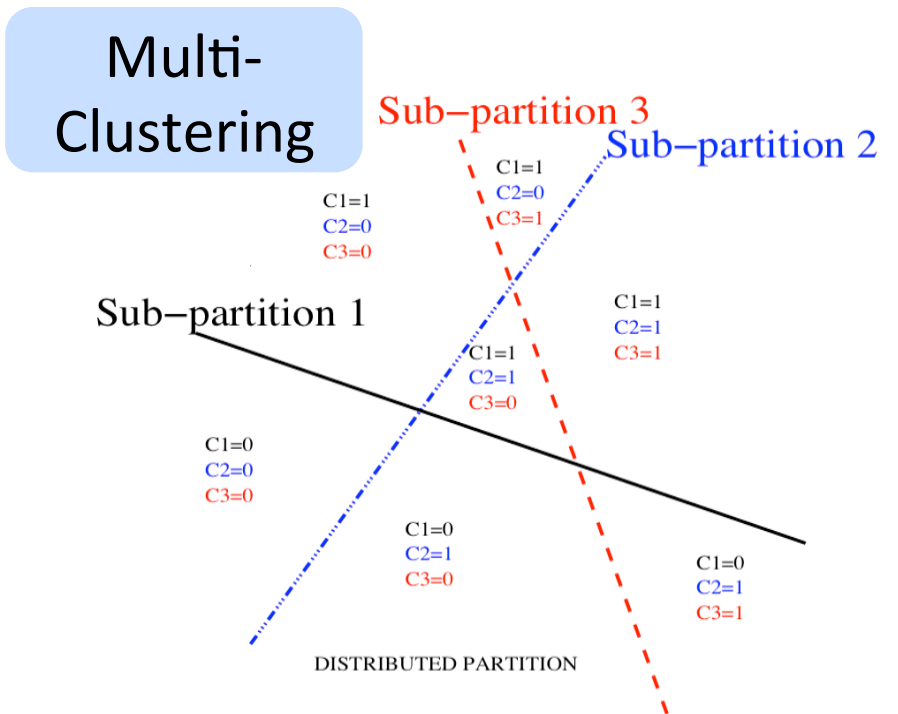
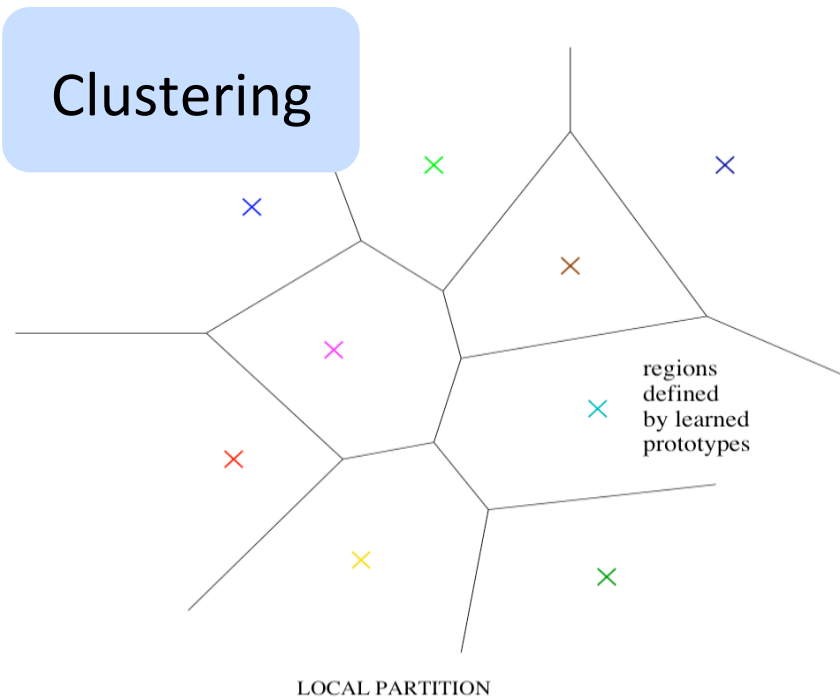
#2 The need for distributed representations

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

Multi-Clustering



#2 The need for distributed representations



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

#3 Unsupervised feature learning

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

But almost all **data is unlabeled**

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

... in about 10^9 seconds

Labels cannot possibly provide enough information

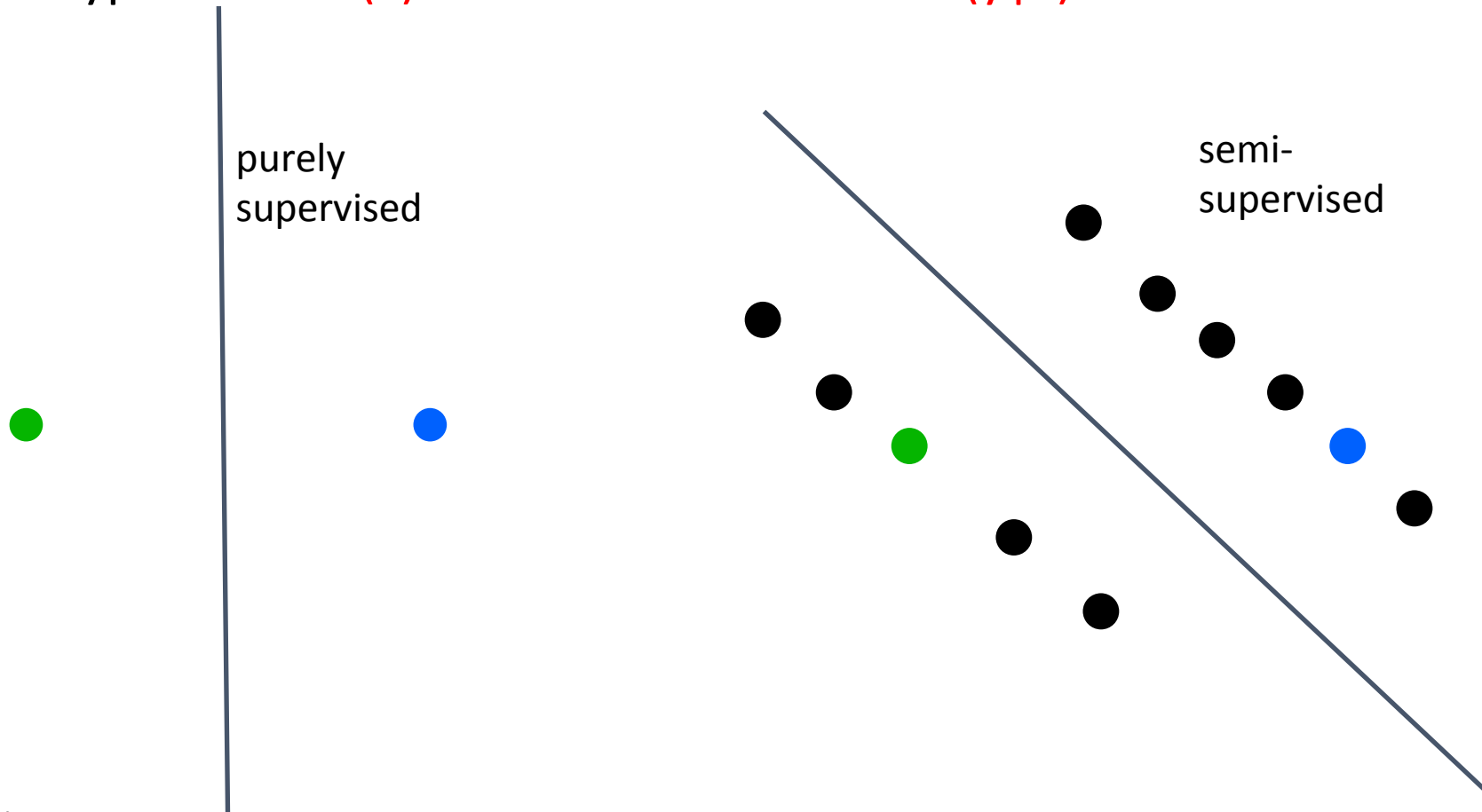
Most information acquired in an **unsupervised** fashion

#3 How do humans generalize from very few examples?

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

#3 Sharing Statistical Strength by Semi-Supervised Learning

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



#4 Learning multiple levels of representation

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

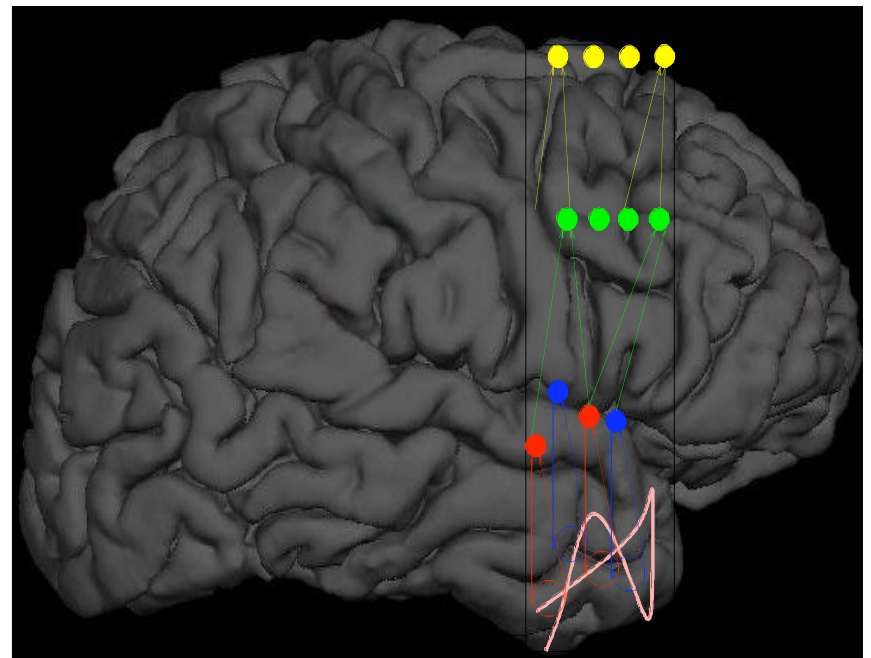
Exponential gain for some families of functions

Biologically inspired learning

Brain has a deep architecture

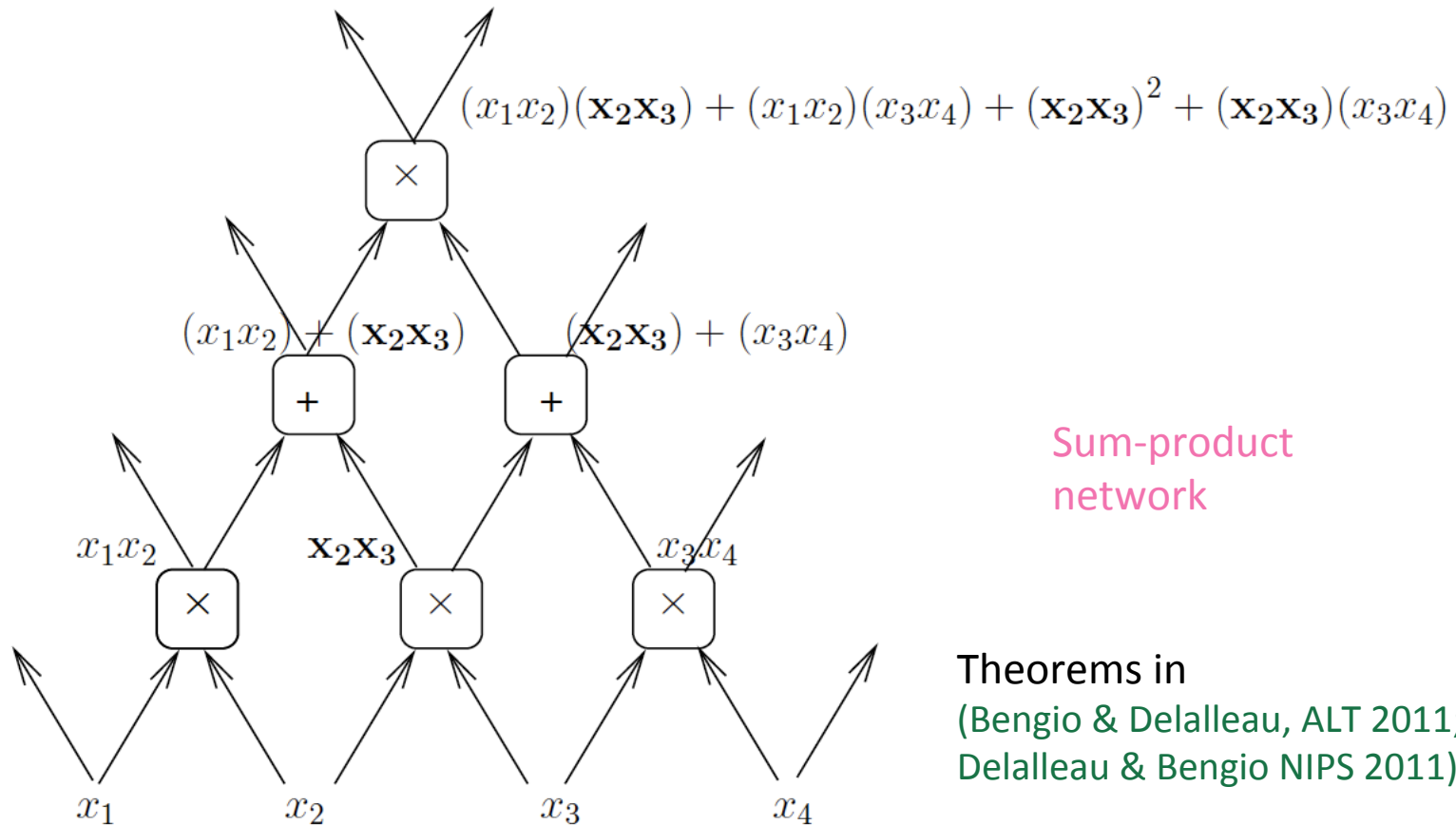
Cortex seems to have a generic learning algorithm

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

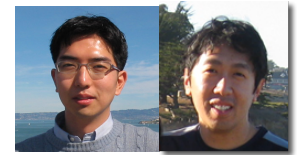


#4 Sharing Components in a Deep Architecture

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

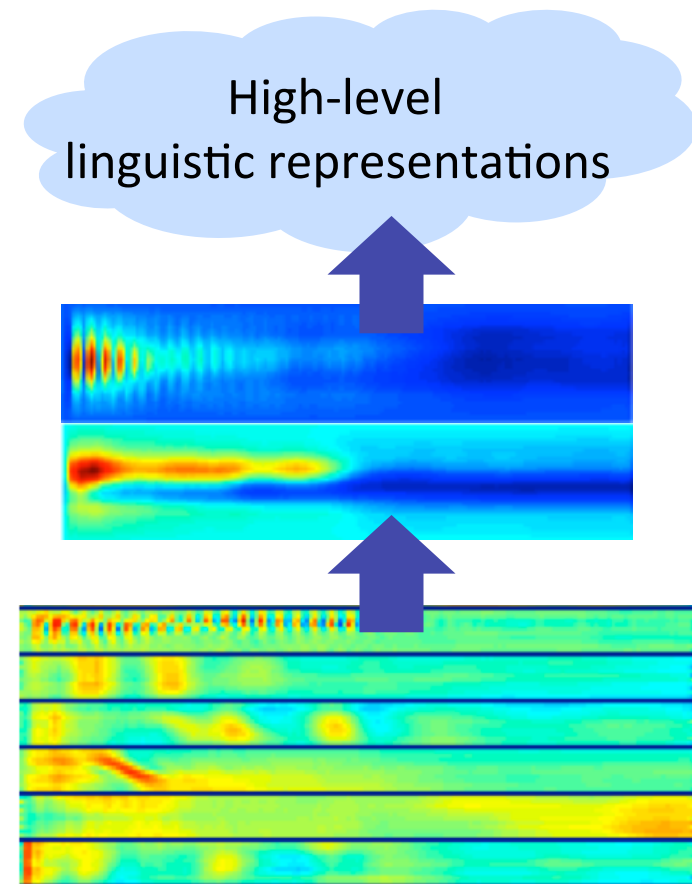
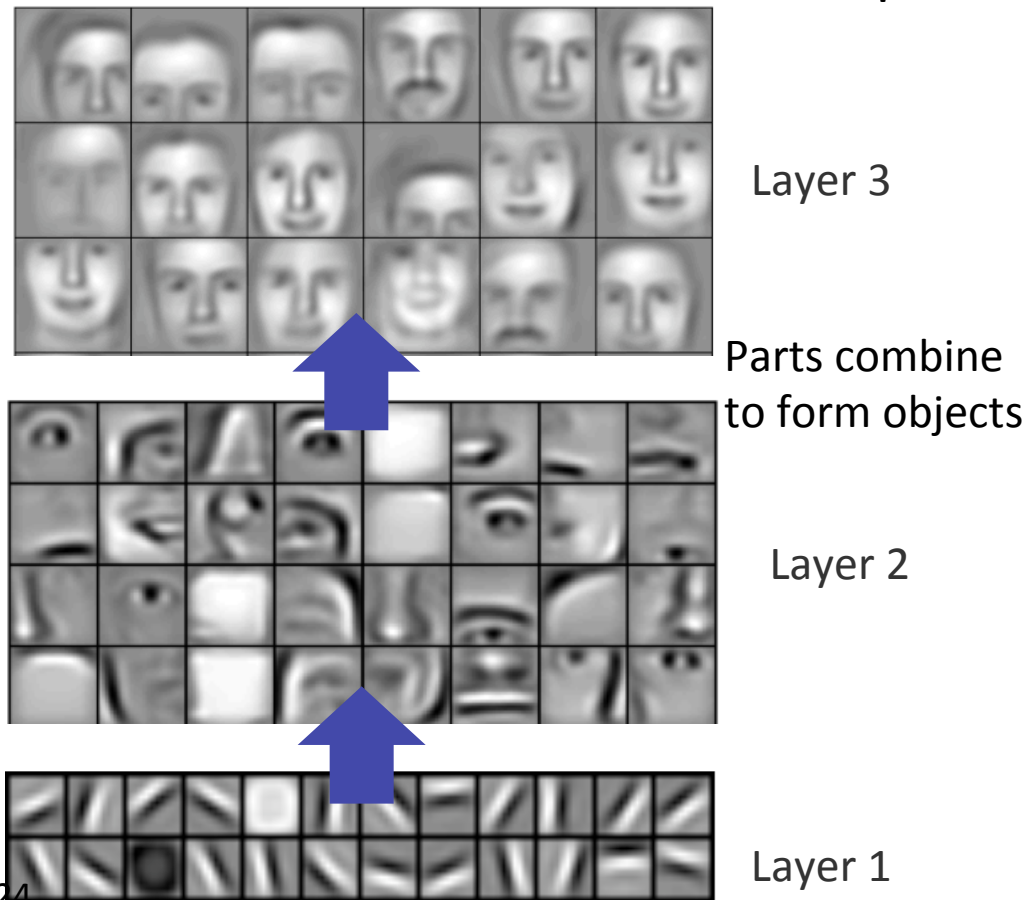


#4 Learning multiple levels of representation



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

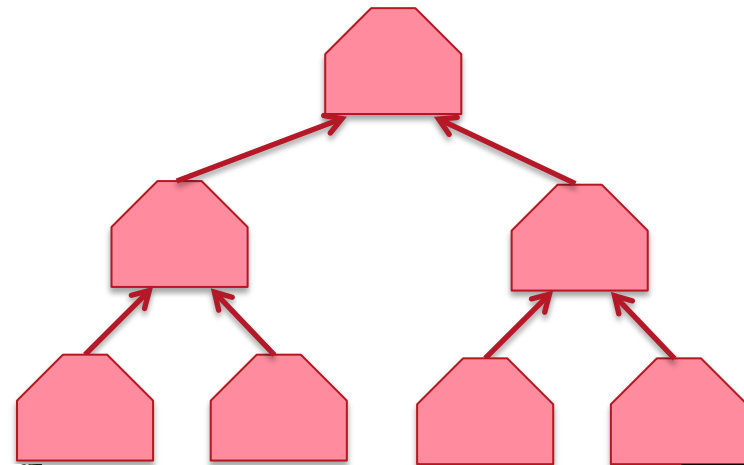
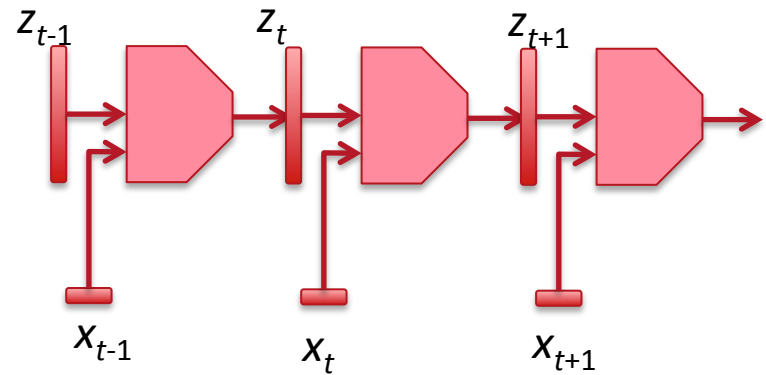
Successive model layers learn deeper intermediate representations



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

#4 Handling the compositionality of human language and thought

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

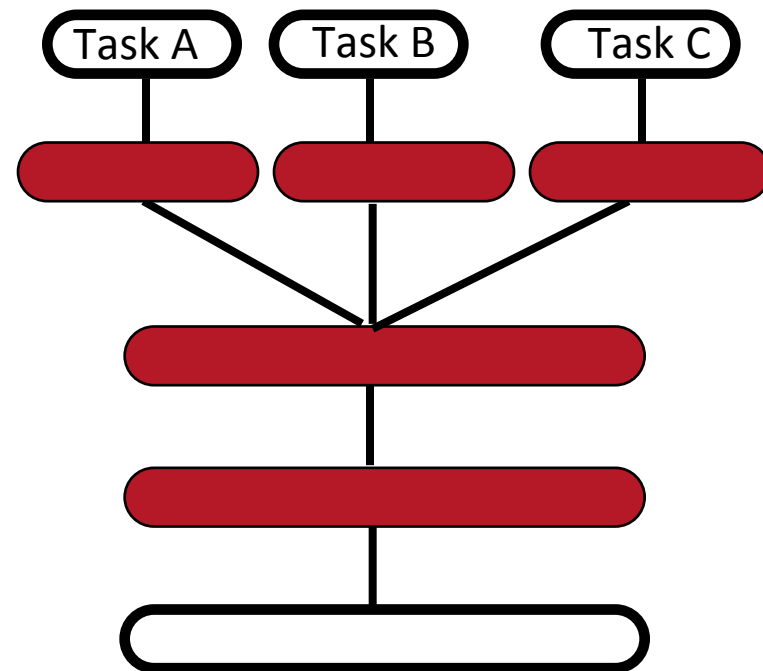


(Bottou 2011, Socher et al 2011)



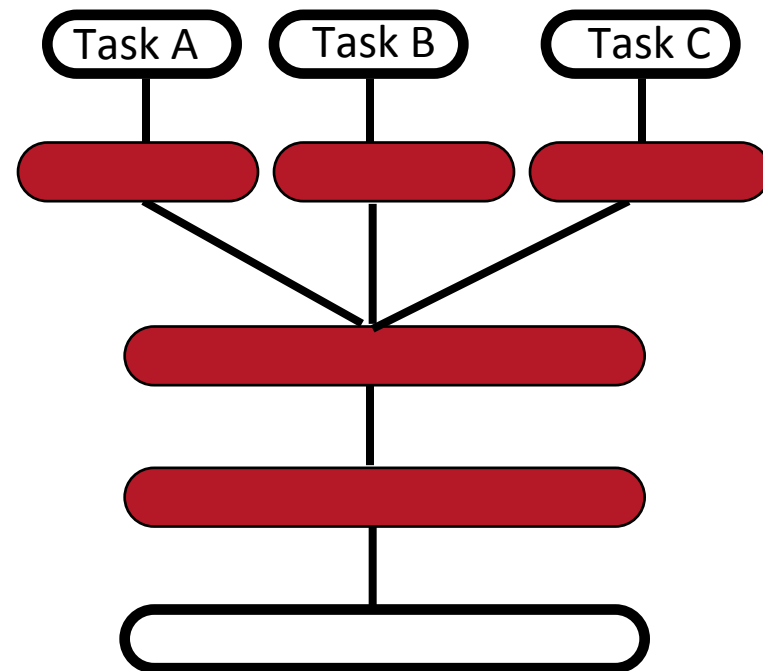
#5 Multi-Task Learning

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



#5 Sharing Statistical Strength

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

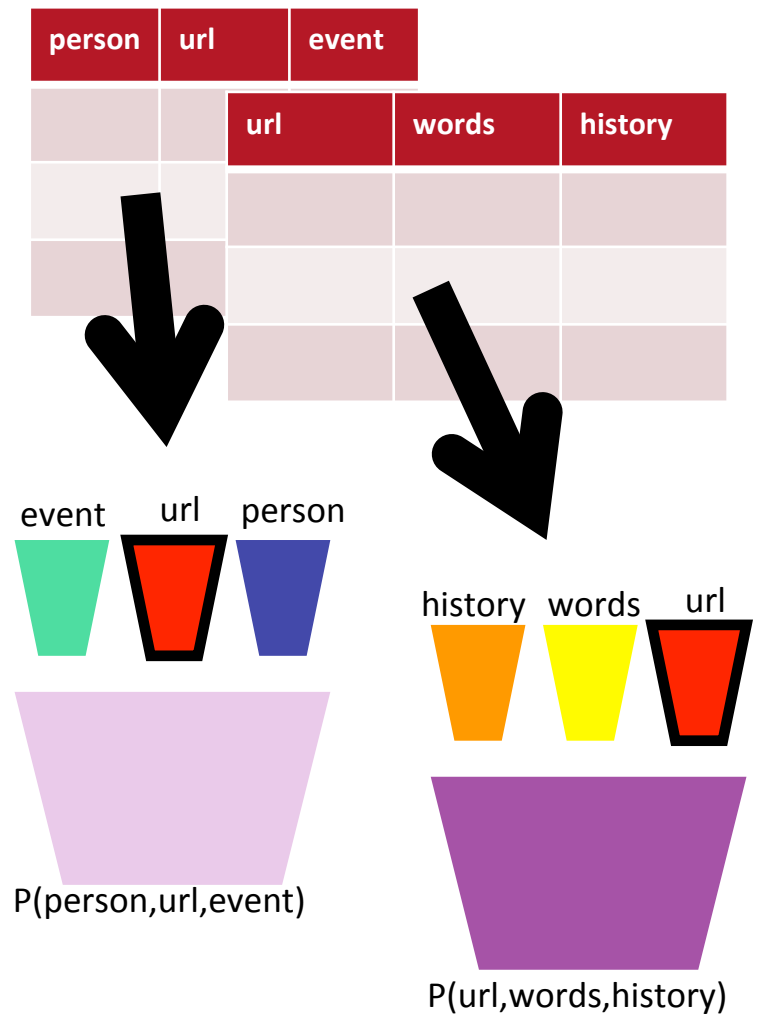


Prior: some shared underlying explanatory factors between tasks

#5 Combining Multiple Sources of Evidence with Shared Representations

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

(Bordes et al AISTATS 2012)



#5 Different object types represented in same space



Google:

S. Bengio, J. Weston & N. Usunier



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



$\Phi_w(\text{DOLPHIN})$

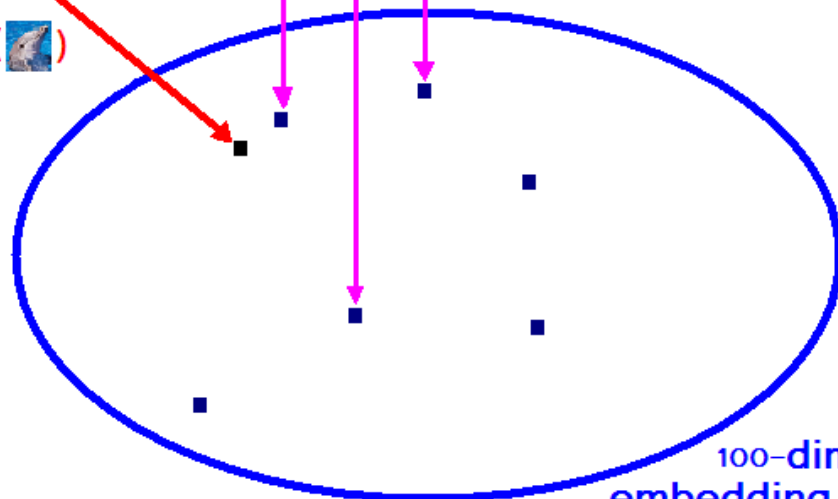
DOLPHIN

OBAMA

EIFFEL TOWER

.....

$\Phi_I(\text{EIFFEL TOWER})$



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

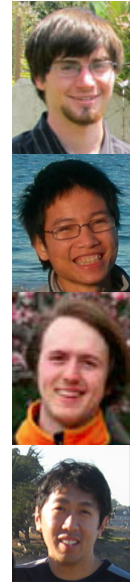
#6 Invariance and Disentangling

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

Few bits of information



Many bits of information



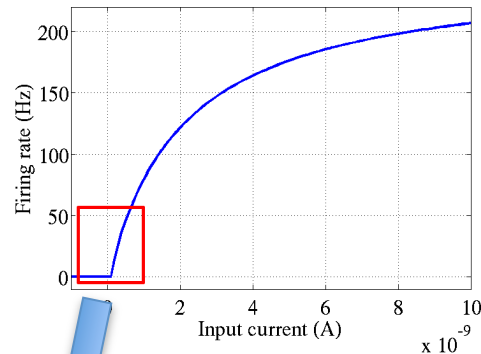
Prior: only few concepts and attributes relevant per example

Deep Sparse Rectifier Neural Networks

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

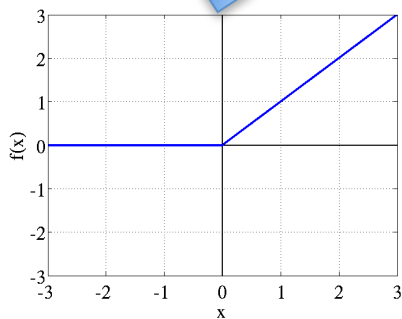
Neuroscience motivations

Leaky integrate-and-fire model



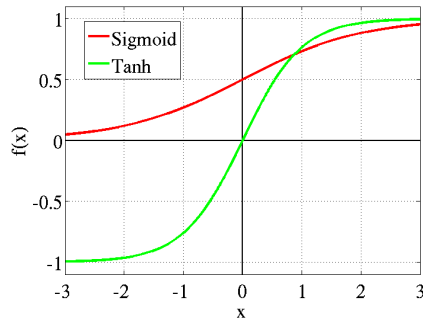
Machine learning motivations

- ➔ Sparse representations
- ➔ Sparse and linear gradients

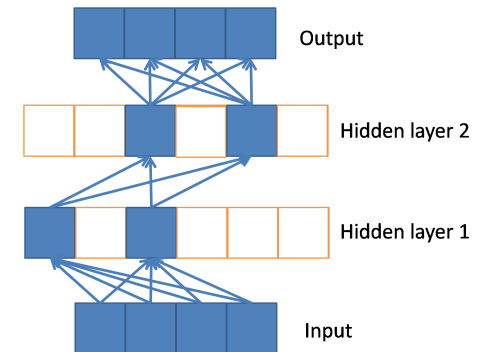


Rectifier

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



Commonly used functions



- ➔ One-sided
- ➔ Real zeros
- ➔ "default" regime at 0

Deep Sparse Rectifier Neural Nets:

Can train deeper supervised nets!

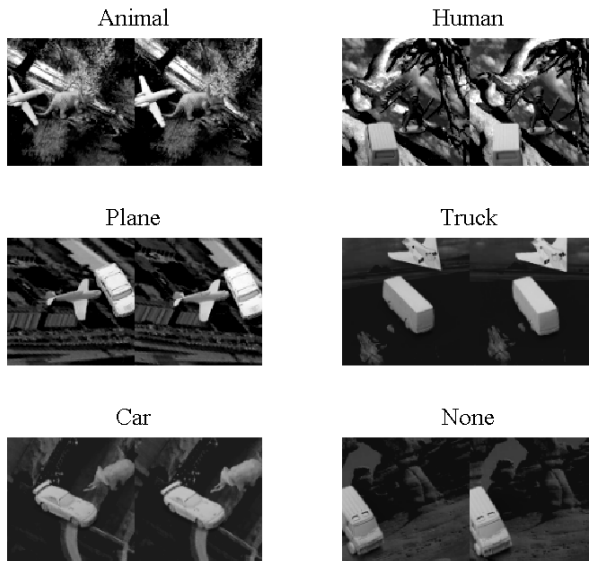
Experiments and results

- ➔ Stacked denoising autoencoder
- ➔ 4 image recognition and 1 sentiment analysis datasets
- ➔ Better generalization than hyperbolic tangent networks
- ➔ Rectifier networks achieve their best performance without needing unsupervised pre-training
- ➔ Unsupervised pre-training is beneficial in the semi-supervised setting

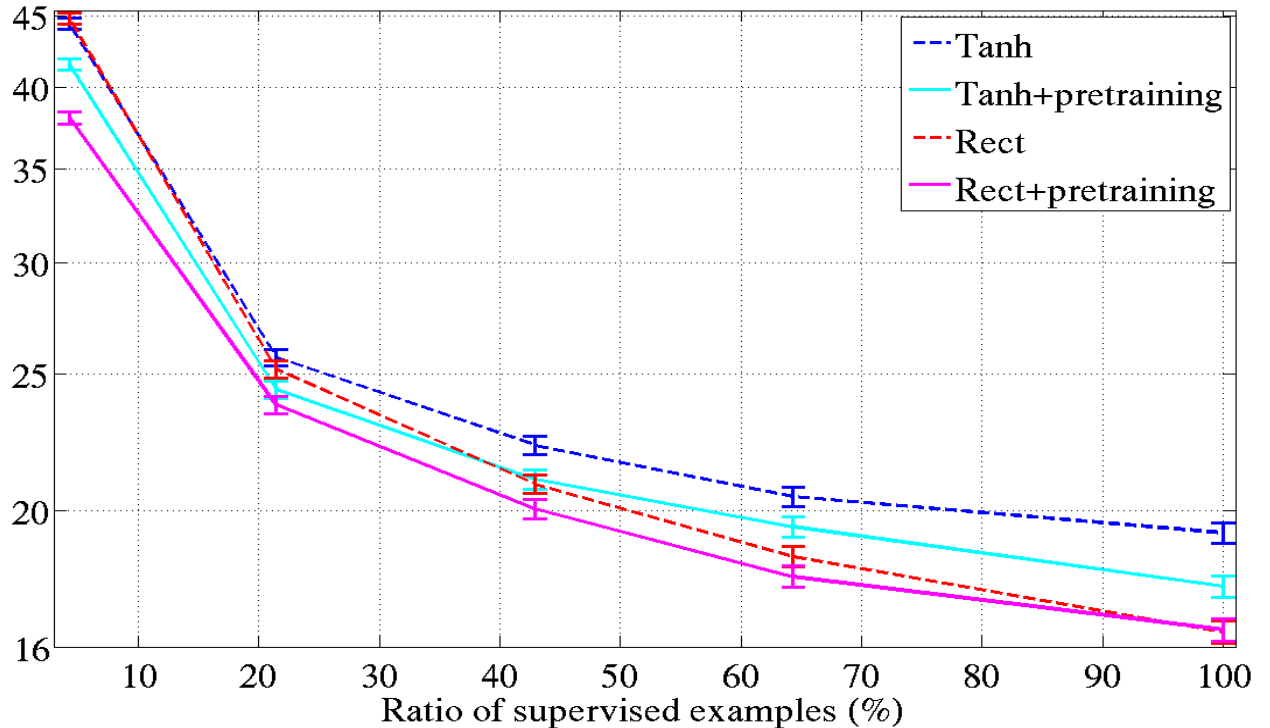
Neuron	MNIST	CIFAR10	NISTP	NORB
With unsupervised pre-training				
Rectifier	1.20%	49.96%	32.86%	16.46%
Tanh	1.16%	50.79%	35.89%	17.66%
Softplus	1.17%	49.52%	33.27%	19.19%
Without unsupervised pre-training				
Rectifier	1.43%	50.86%	32.64%	16.40%
Tanh	1.57%	52.62%	36.46%	19.29%
Softplus	1.77%	53.20%	35.48%	17.68%



NISTP



NORB



Temporal Coherence and Scales

- One of the hints from nature about different explanatory factors:
 - Rapidly changing factors (often noise)
 - Slowly changing (generally more abstract)
 - Different factors at different time scales
- We should 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

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

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

Exploiting compositionality gives an exponential gain in representational power

Distributed representations / embeddings: **feature learning**

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

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

Bypassing the curse by sharing statistical strength

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

Why now?

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

Before 2006 training deep architectures was **unsuccessful**

(except for convolutional neural nets when used by people who speak French)

What has changed?

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

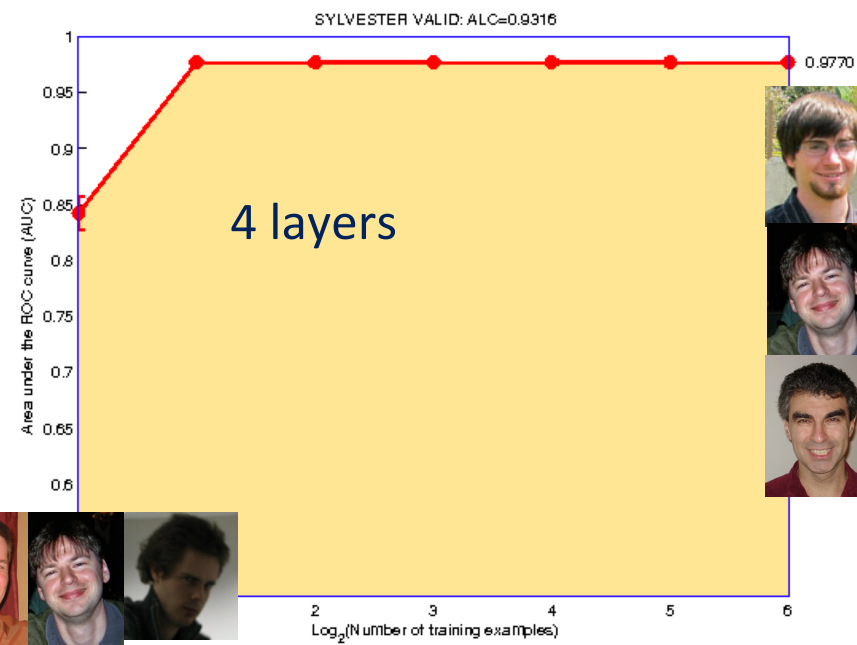
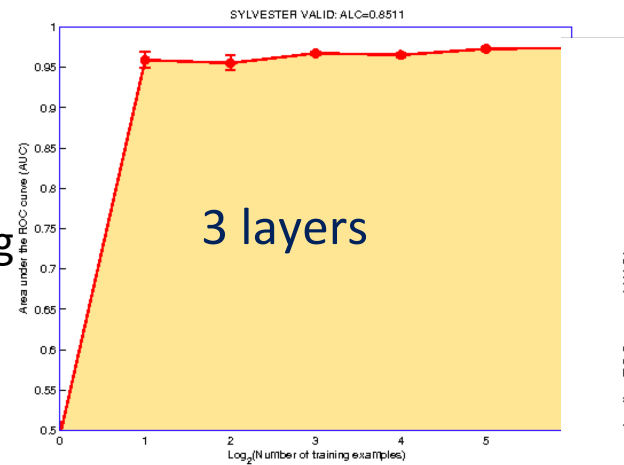
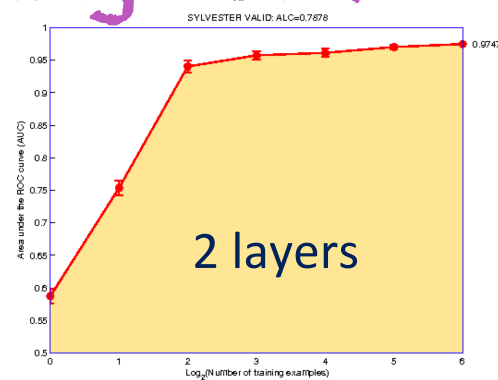
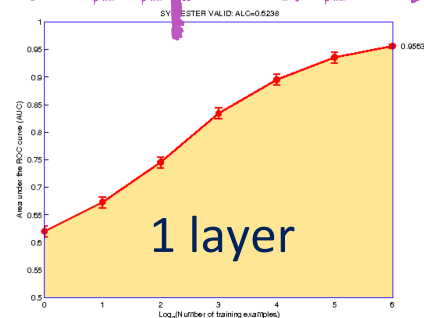
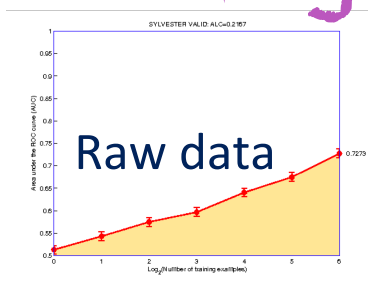
Major Breakthrough in 2006



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



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



NIPS'2011
Transfer Learning
Challenge
Paper:
ICML'2012

ICML'2011
workshop on
Unsup. &
Transfer Learning



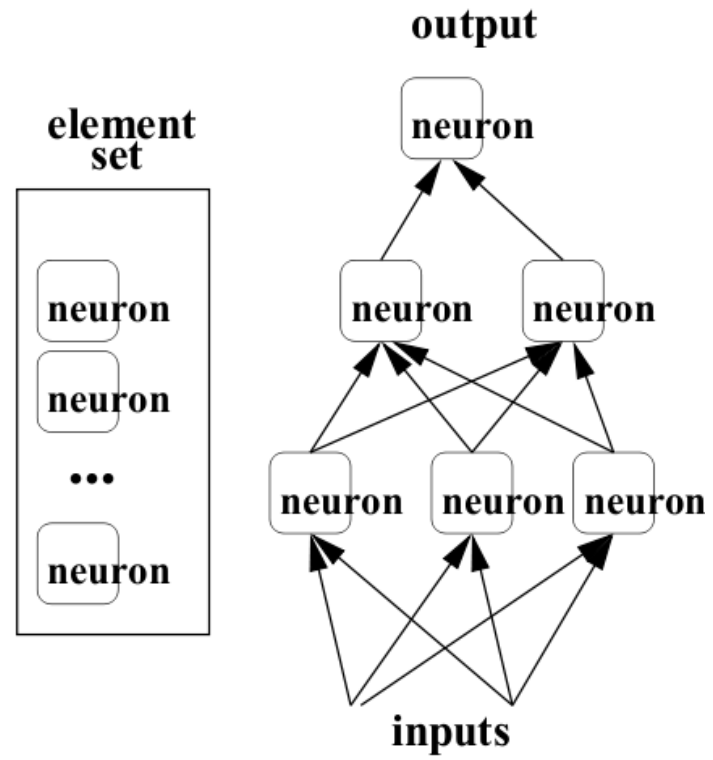
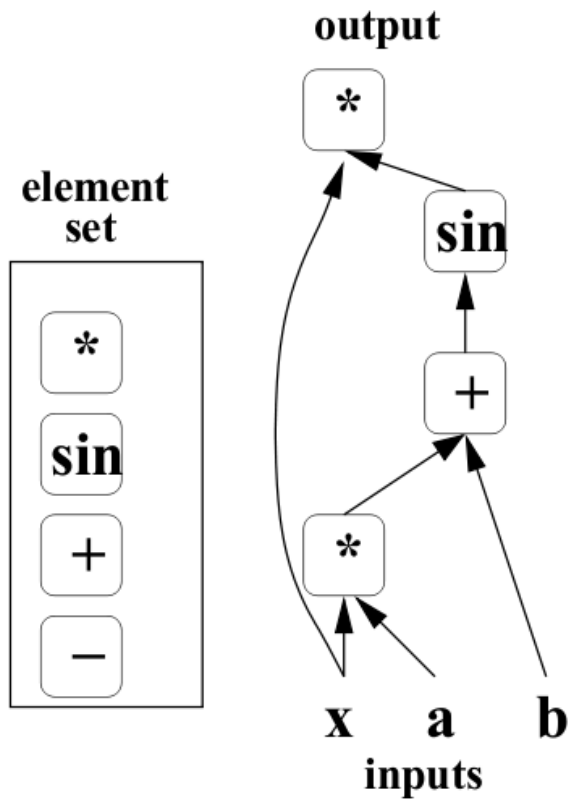
More Successful Applications

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

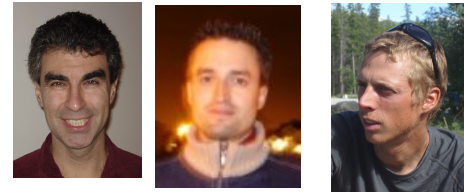


More about depth

Architecture Depth



Deep Architectures are More Expressive



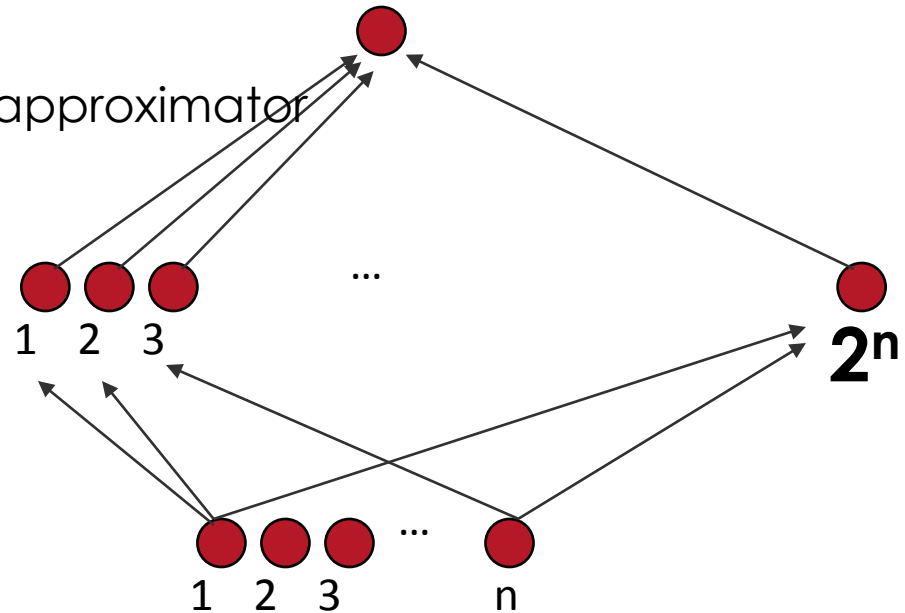
Theoretical arguments:

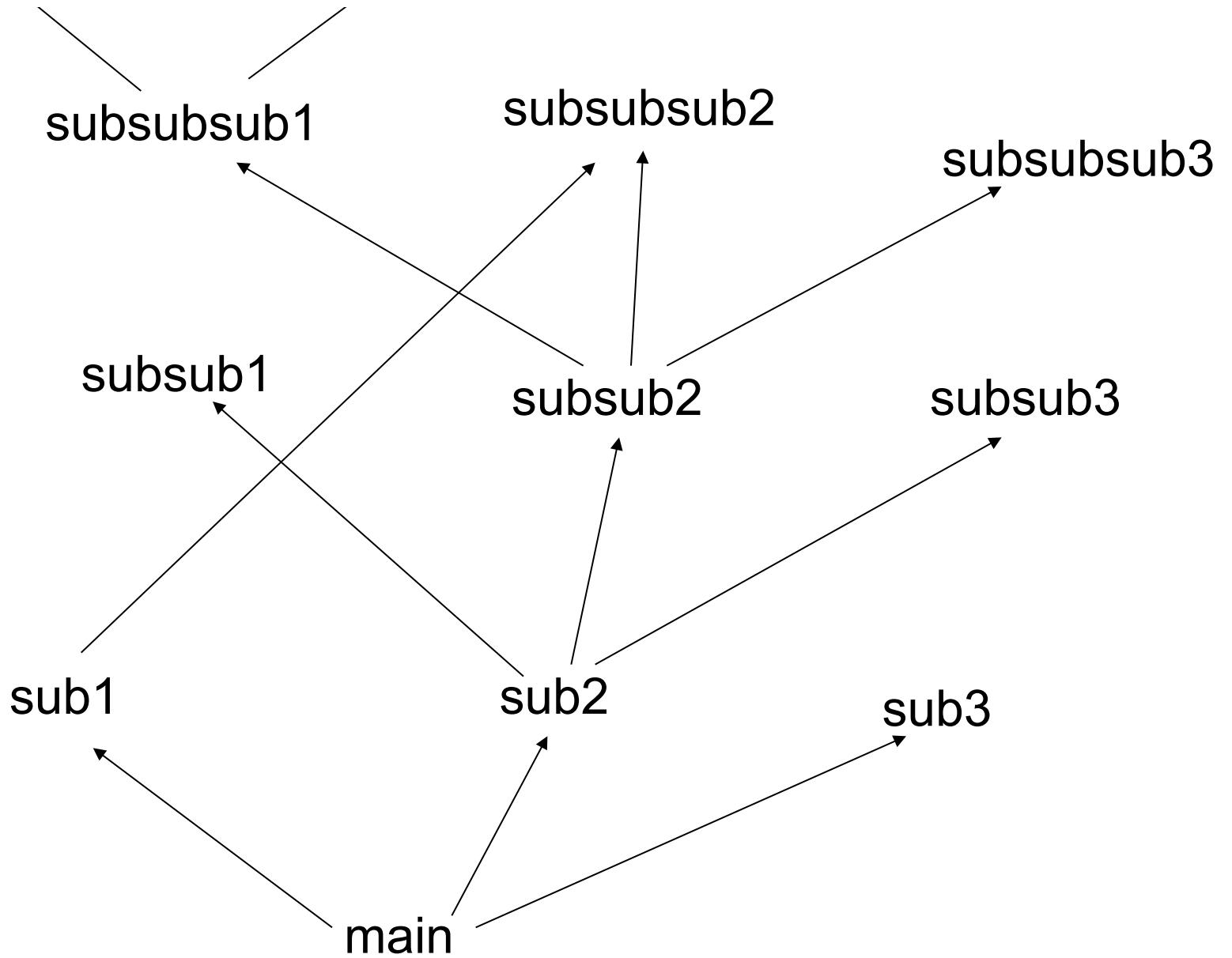
2 layers of $\left\{ \begin{array}{l} \text{Logic gates} \\ \text{Formal neurons} \\ \text{RBF units} \end{array} \right.$ = universal approximator

RBMs & auto-encoders = universal approximator

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

Functions compactly represented with k layers may require exponential size with 2 layers

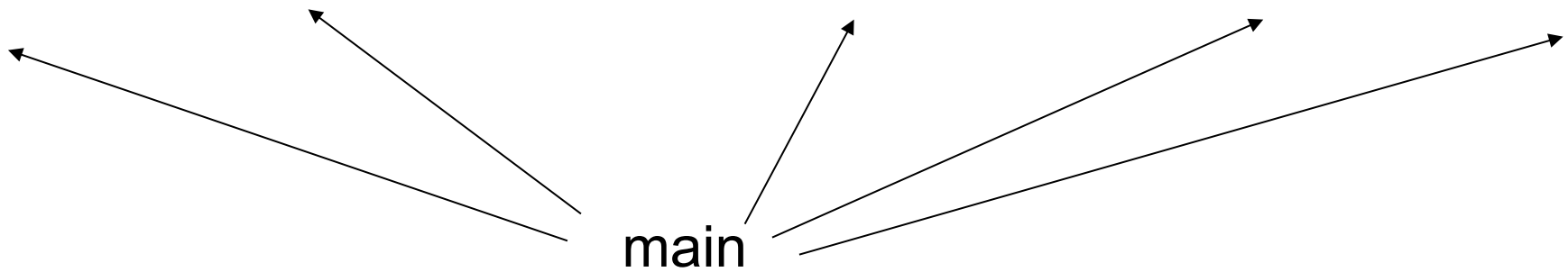




“Deep” computer program

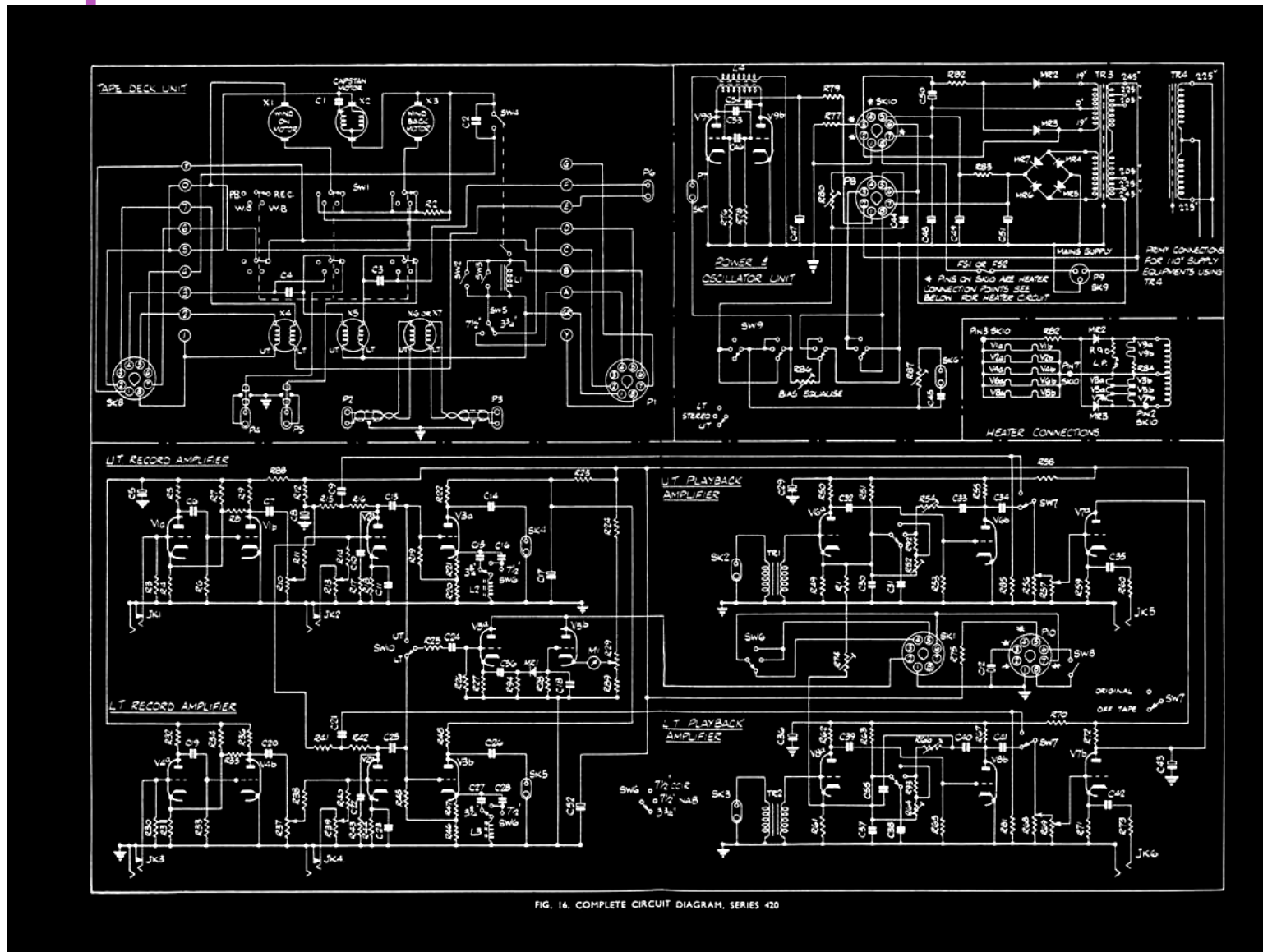
subroutine1 includes
subsub1 code and
subsub2 code and
subsubsub1 code

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

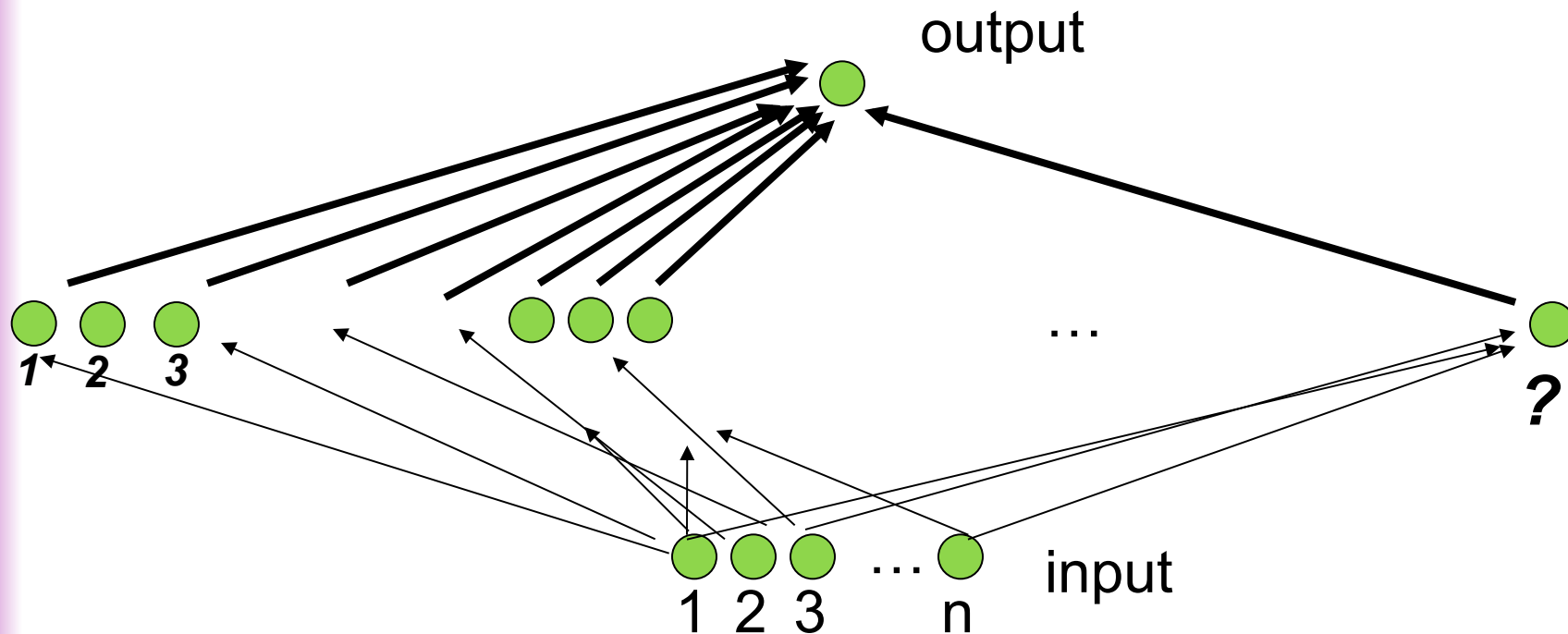


“Shallow” computer program

“Deep” circuit

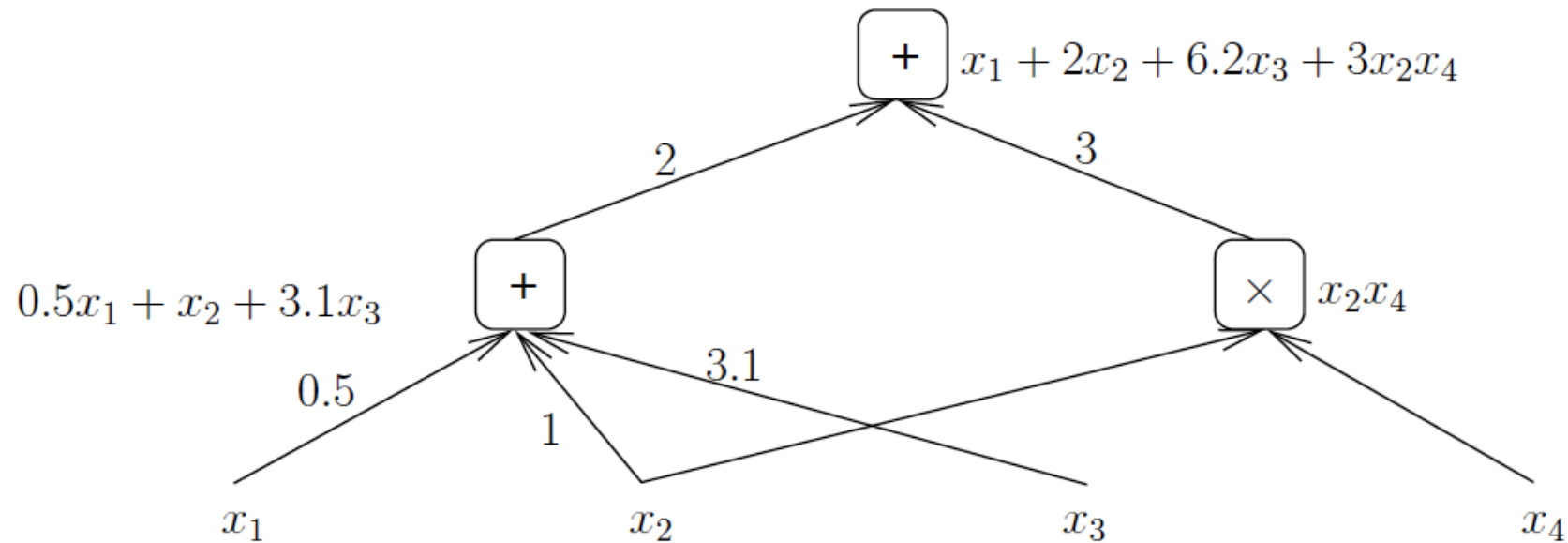


"Shallow" circuit



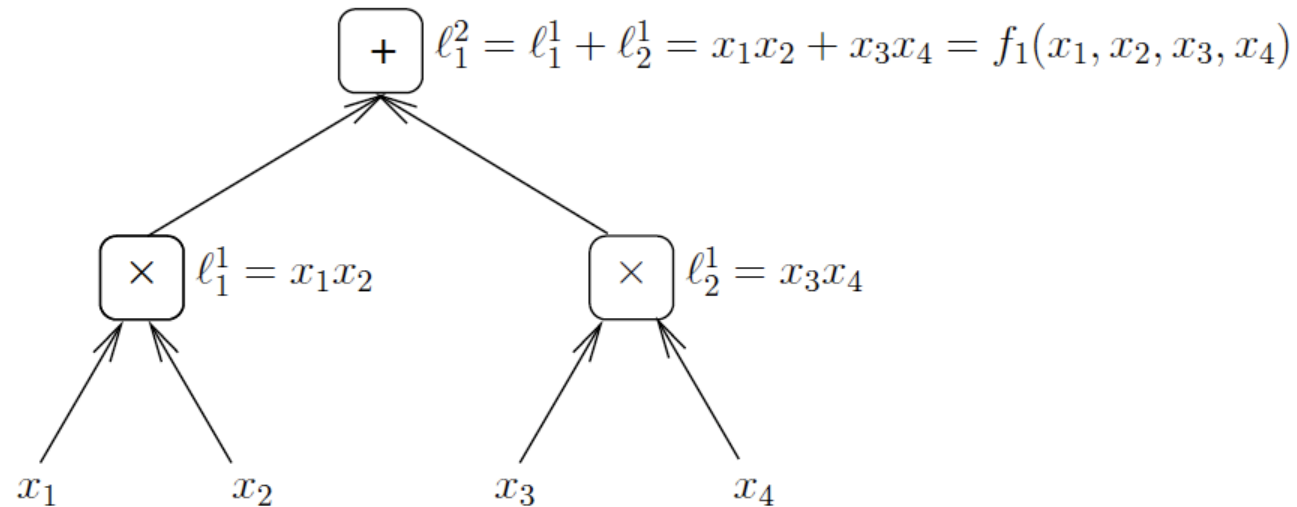
Falsely reassuring theorems: one can approximate any reasonable (smooth, boolean, etc.) function with a 2-layer architecture

Sum-Product Networks



Depth 2 suffices to represent any finite polynomial (sum of products)
(Poon & Domingos 2010) use deep sum-product networks to efficiently parametrize partition functions

Polynomials that Need Depth

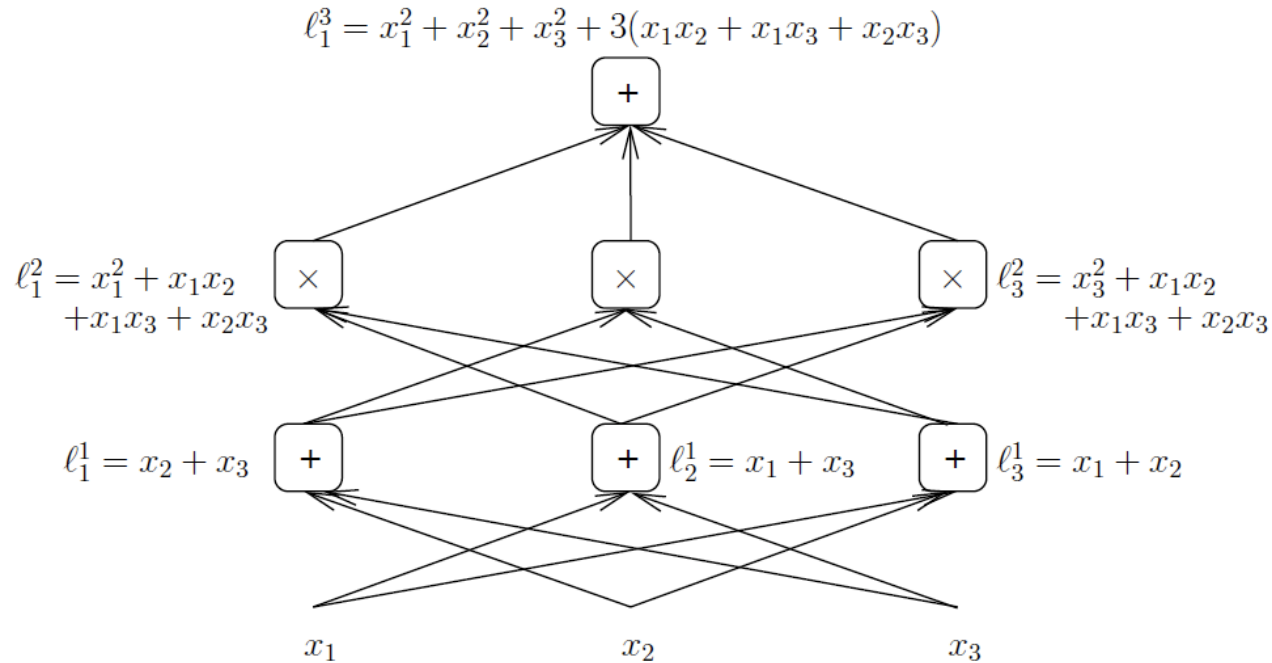


- $2i$ layers and $n = 4^i$ input variables
- alternate additive and multiplicative units
- unit l_j^k takes as inputs l_{2j-1}^{k-1} and l_{2j}^{k-1}

Need $O(n)$ nodes with depth $\log(n)$ circuit

Need $O(2^{\vee n})$ nodes with depth-2 circuit

More Polynomials that Need Depth



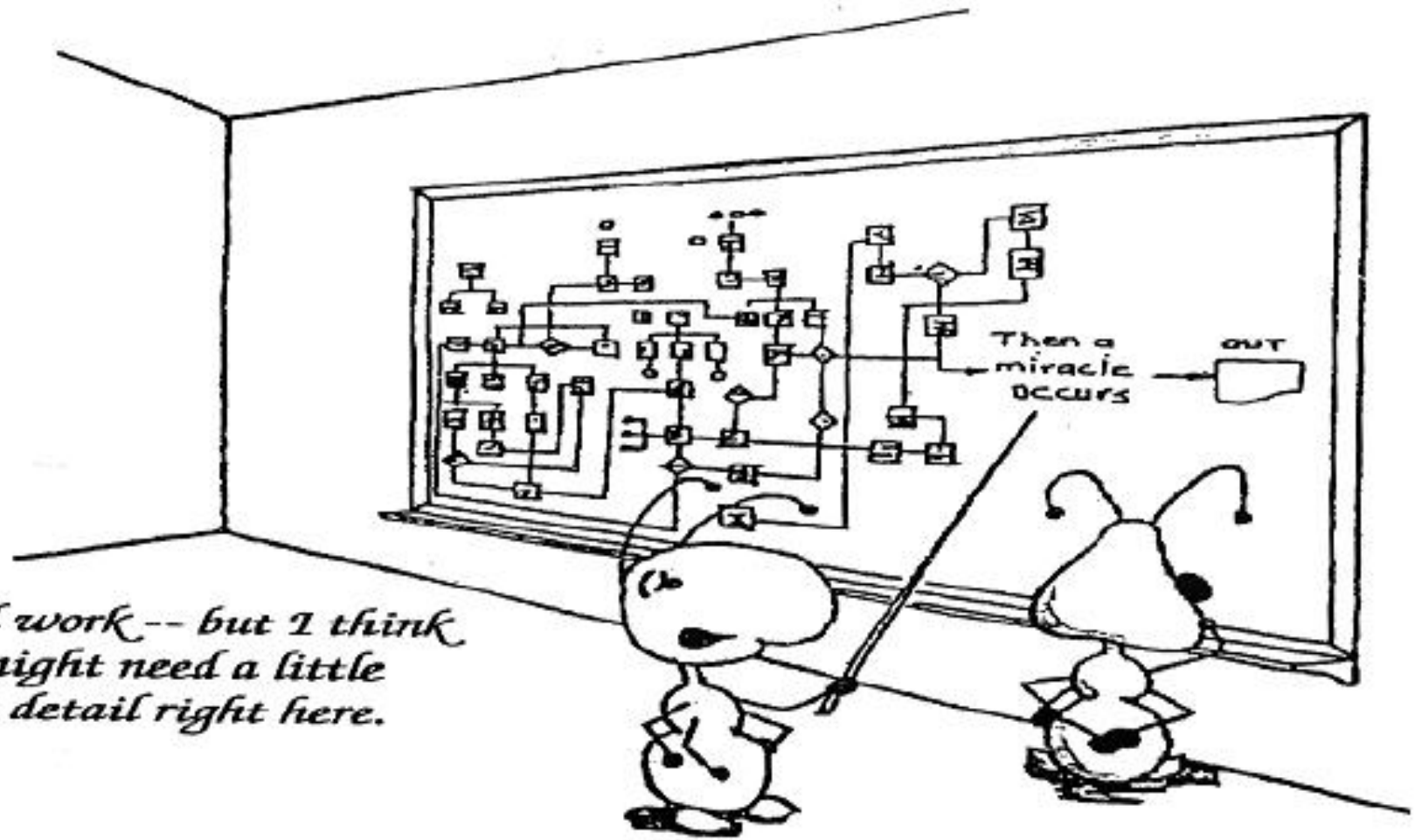
- $2i + 1$ layers and n variables (n independent of i)
- alternate multiplicative and additive units
- unit ℓ_j^k takes as inputs $\{\ell_m^{k-1} | m \neq j\}$

More Deep Theory

Poly-logarithmic Independence Fools Bounded-Depth Boolean Circuits

Braverman, CACM 54(4), April 2011.

If all marginals of the input distribution involving at most k variables are uniform, higher depth makes it exponentially easier to distinguish the joint from the uniform.



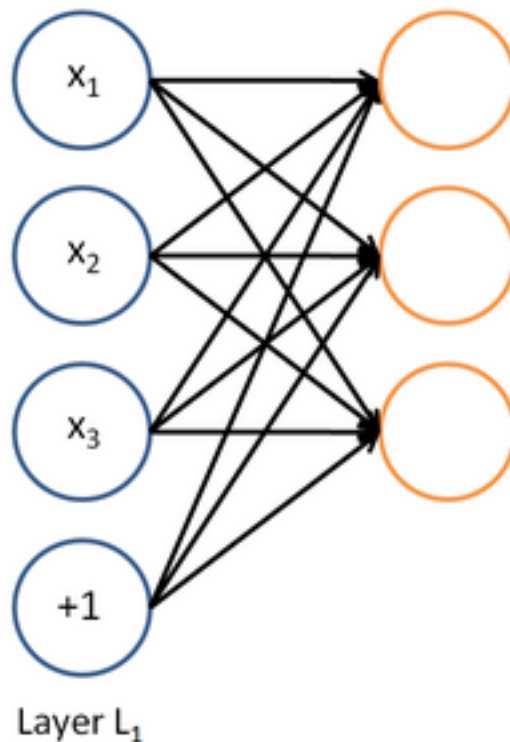
Good work-- but I think we might need a little more detail right here.

Part 2

Representation Learning Algorithms

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

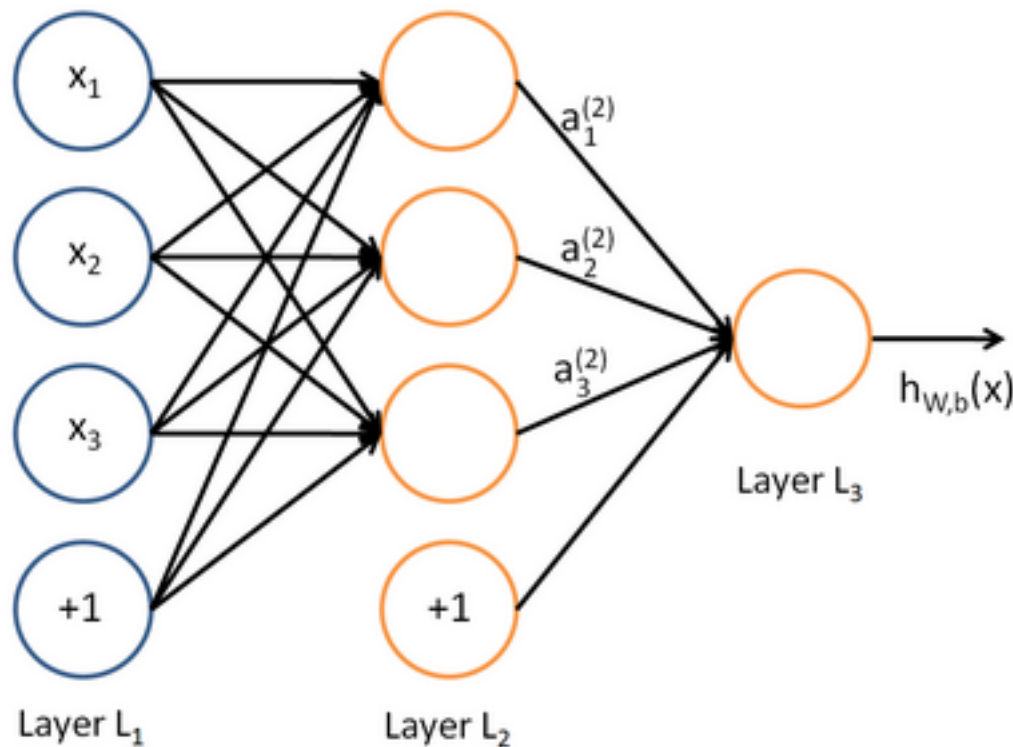
If we feed a vector of inputs through a bunch of logistic regression functions, then we get a vector of outputs



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

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

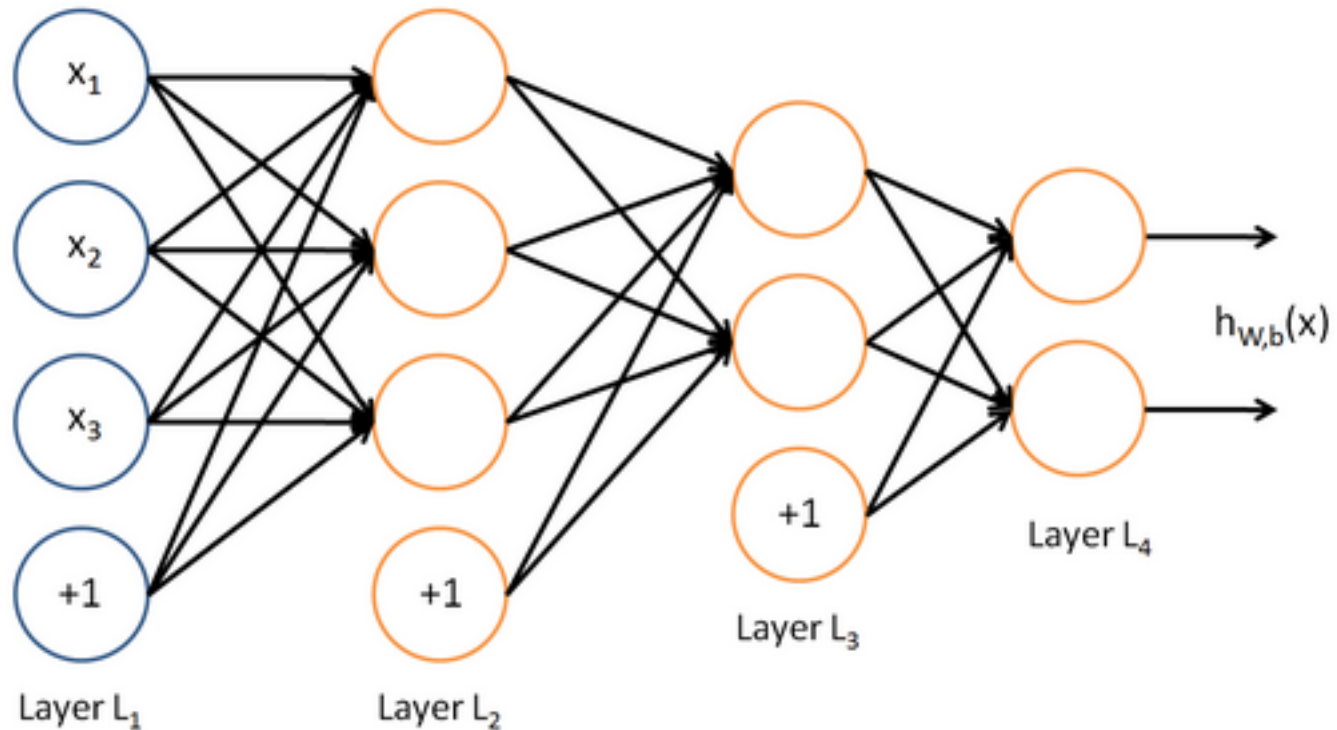
... which we can feed into another logistic regression function



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

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

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



Back-Prop

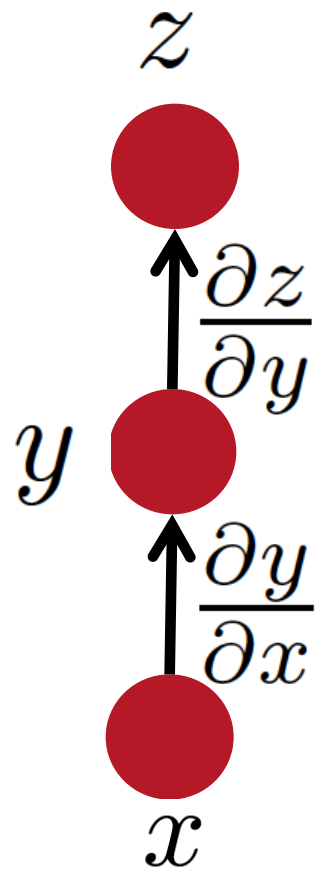
- Compute gradient of example-wise loss wrt parameters

- Simply applying the derivative chain rule wisely

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

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

Simple Chain Rule



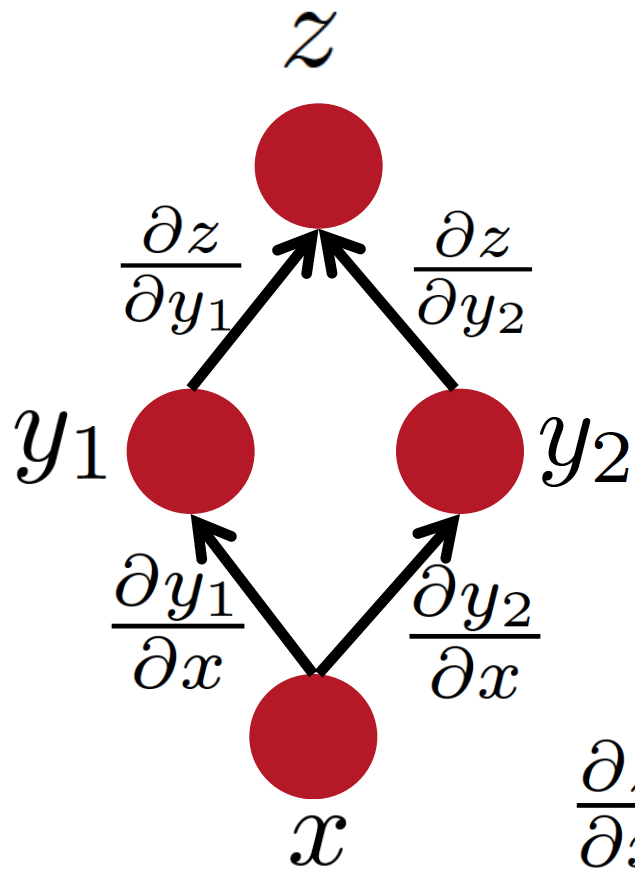
$$\Delta z = \frac{\partial z}{\partial y} \Delta y$$

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

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

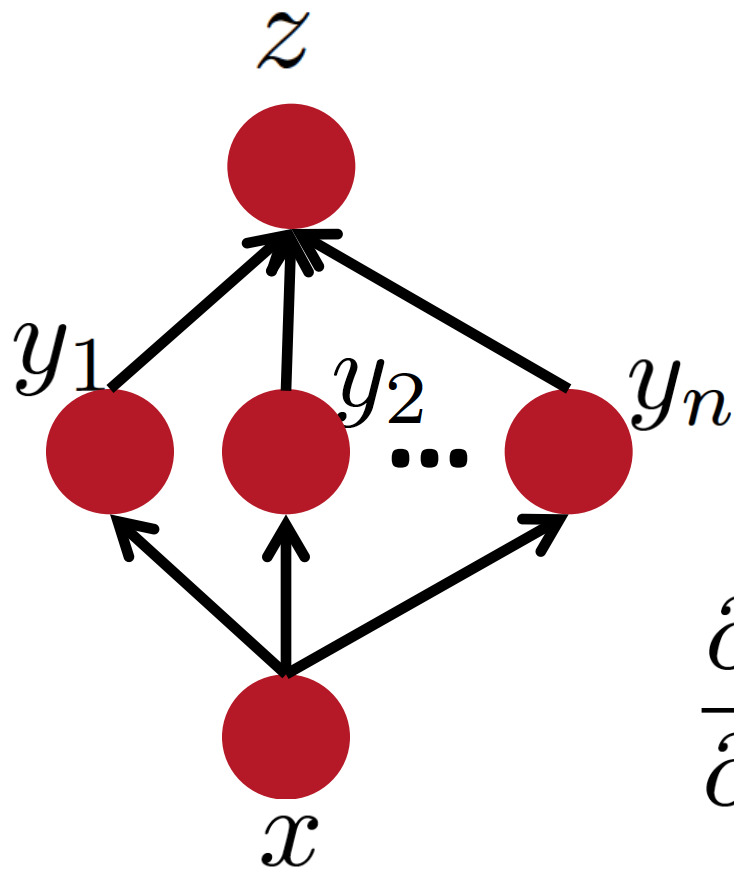
$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \frac{\partial y}{\partial x}$$

Multiple Paths Chain Rule



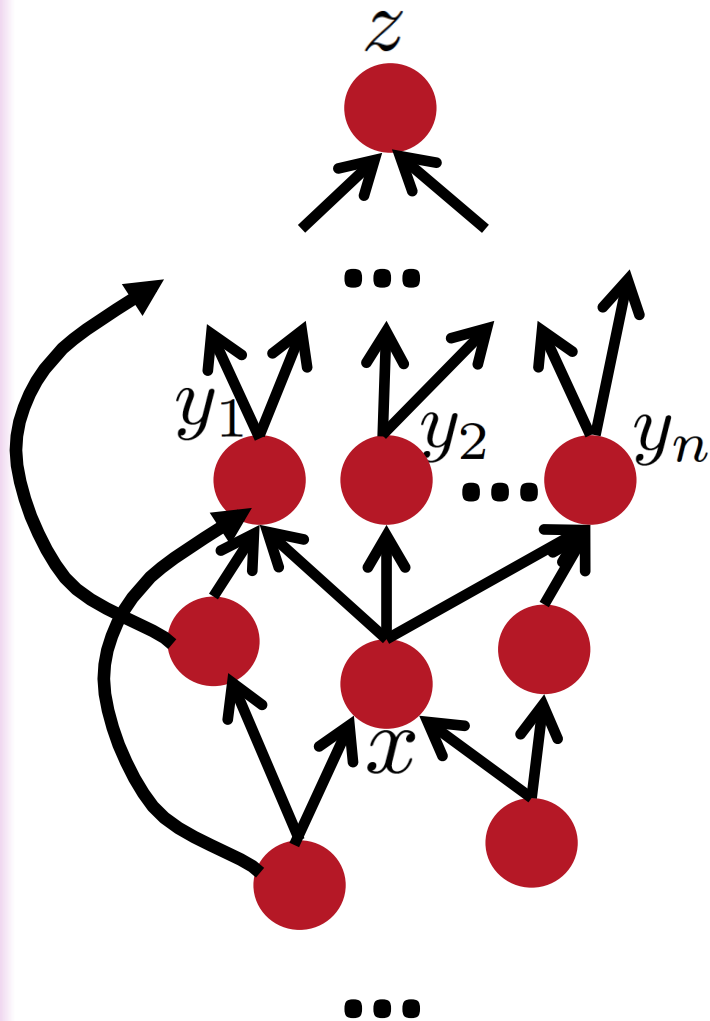
$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y_1} \frac{\partial y_1}{\partial x} + \frac{\partial z}{\partial y_2} \frac{\partial y_2}{\partial x}$$

Multiple Paths Chain Rule - General



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

Chain Rule in Flow Graph

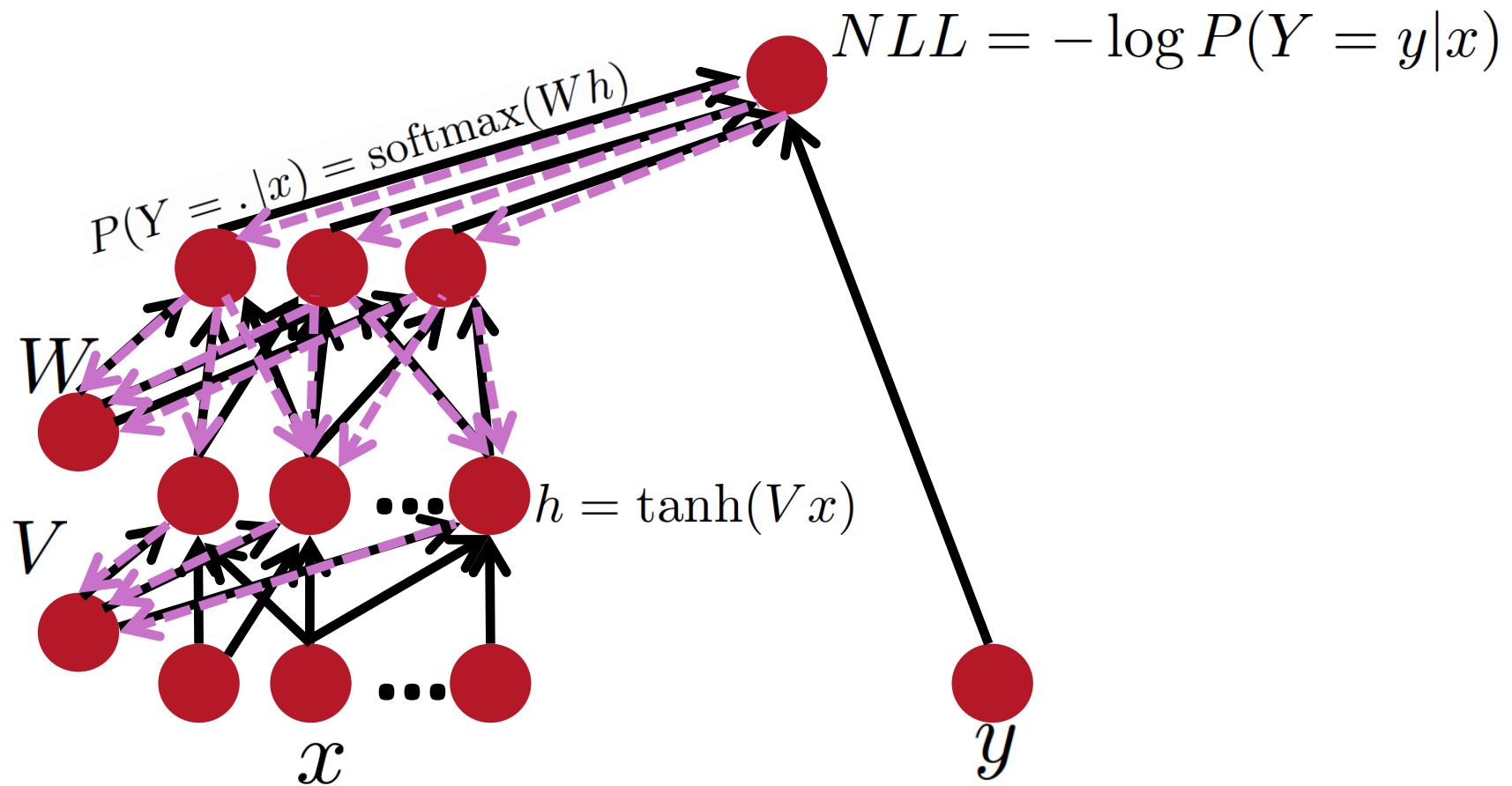


Flow graph: any directed acyclic graph
node = computation result
arc = computation dependency

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

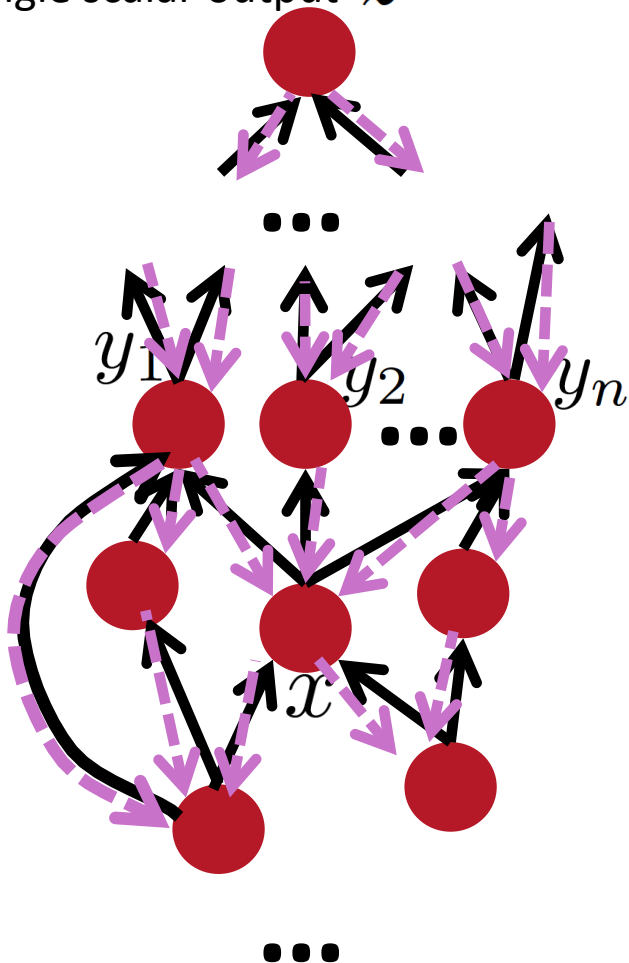
$$\frac{\partial z}{\partial x} = \sum_{i=1}^n \frac{\partial z}{\partial y_i} \frac{\partial y_i}{\partial x}$$

Back-Prop in Multi-Layer Net



Back-Prop in General Flow Graph

Single scalar output z



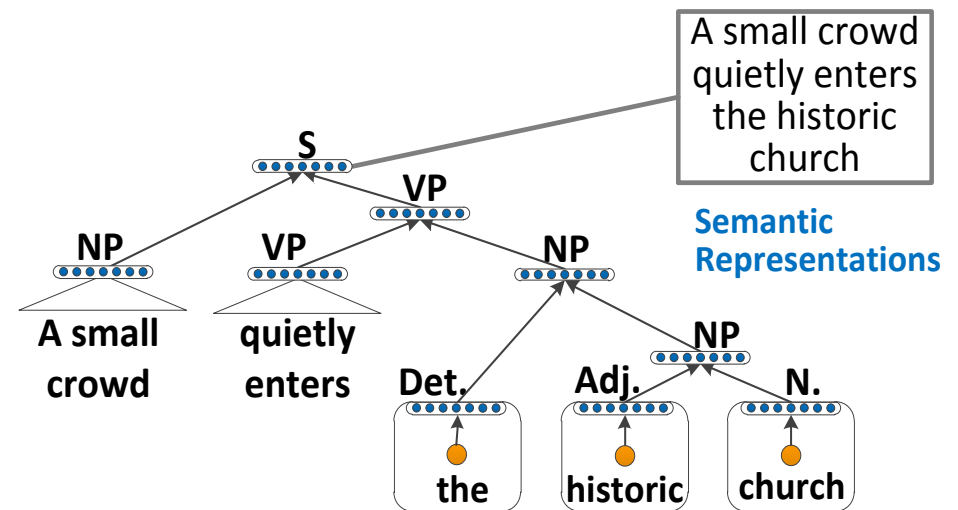
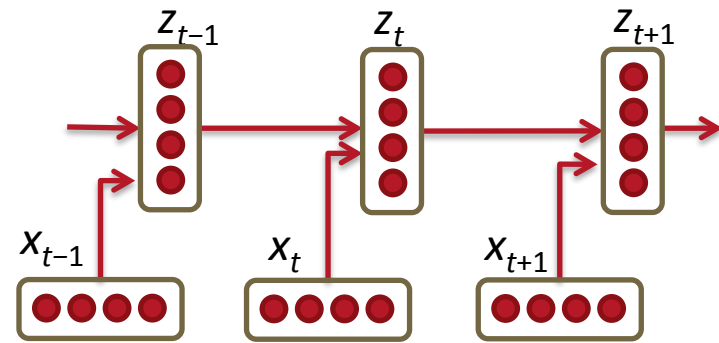
1. Fprop: visit nodes in topo-sort order
 - Compute value of node given predecessors
2. Bprop:
 - initialize output gradient = 1
 - visit nodes in reverse order:
 - Compute gradient wrt each node using gradient wrt successors

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

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

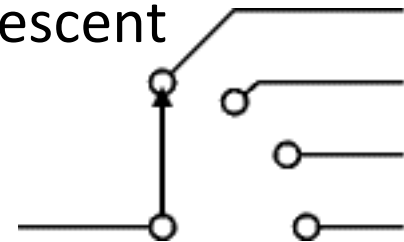
Back-Prop in Recurrent & Recursive Nets

- 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

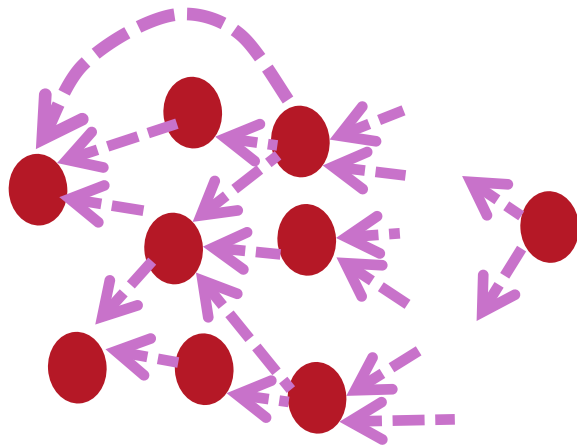
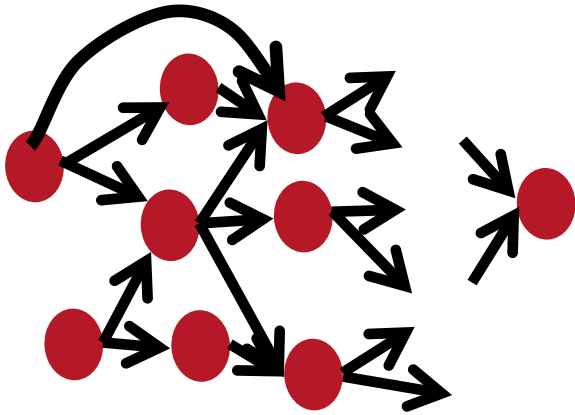


Backpropagation Through Structure

- 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

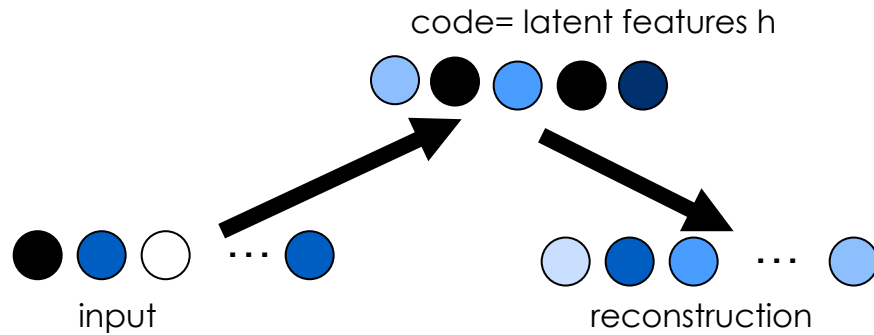


- The gradient computation can be automatically inferred from the symbolic expression of the fprop.
- Each node type needs to know how to compute its output and how to compute the gradient wrt its inputs given the gradient wrt its output.
- Easy and fast prototyping

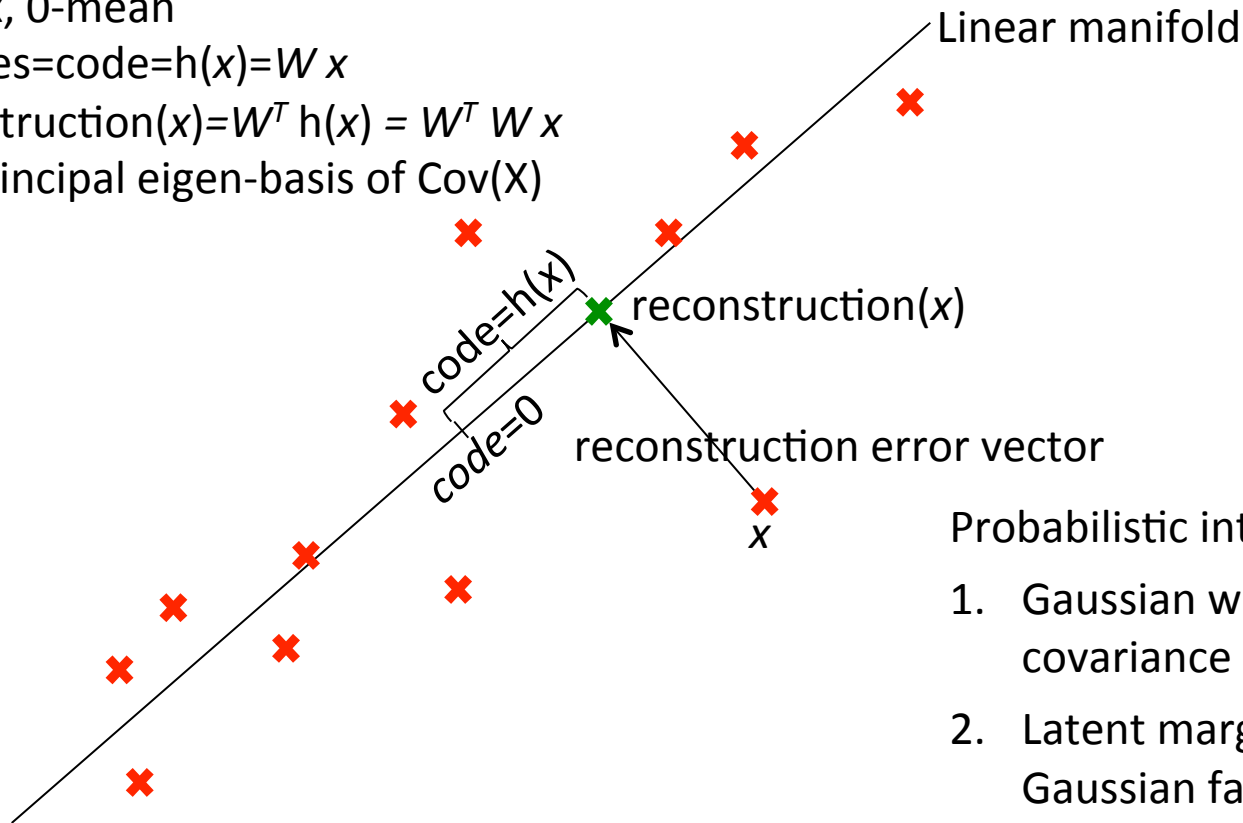
Distributed Representations and Neural
Nets: How to do unsupervised training?

PCA

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



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

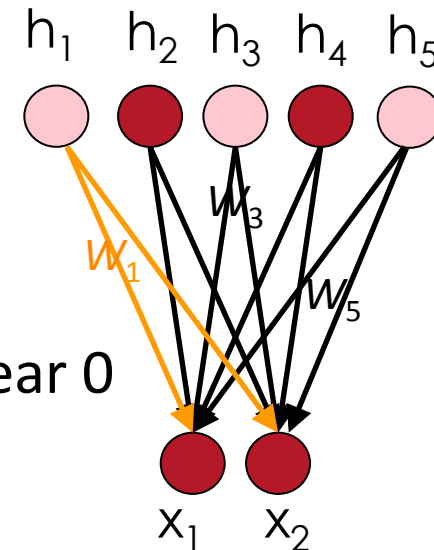


Probabilistic interpretations:

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

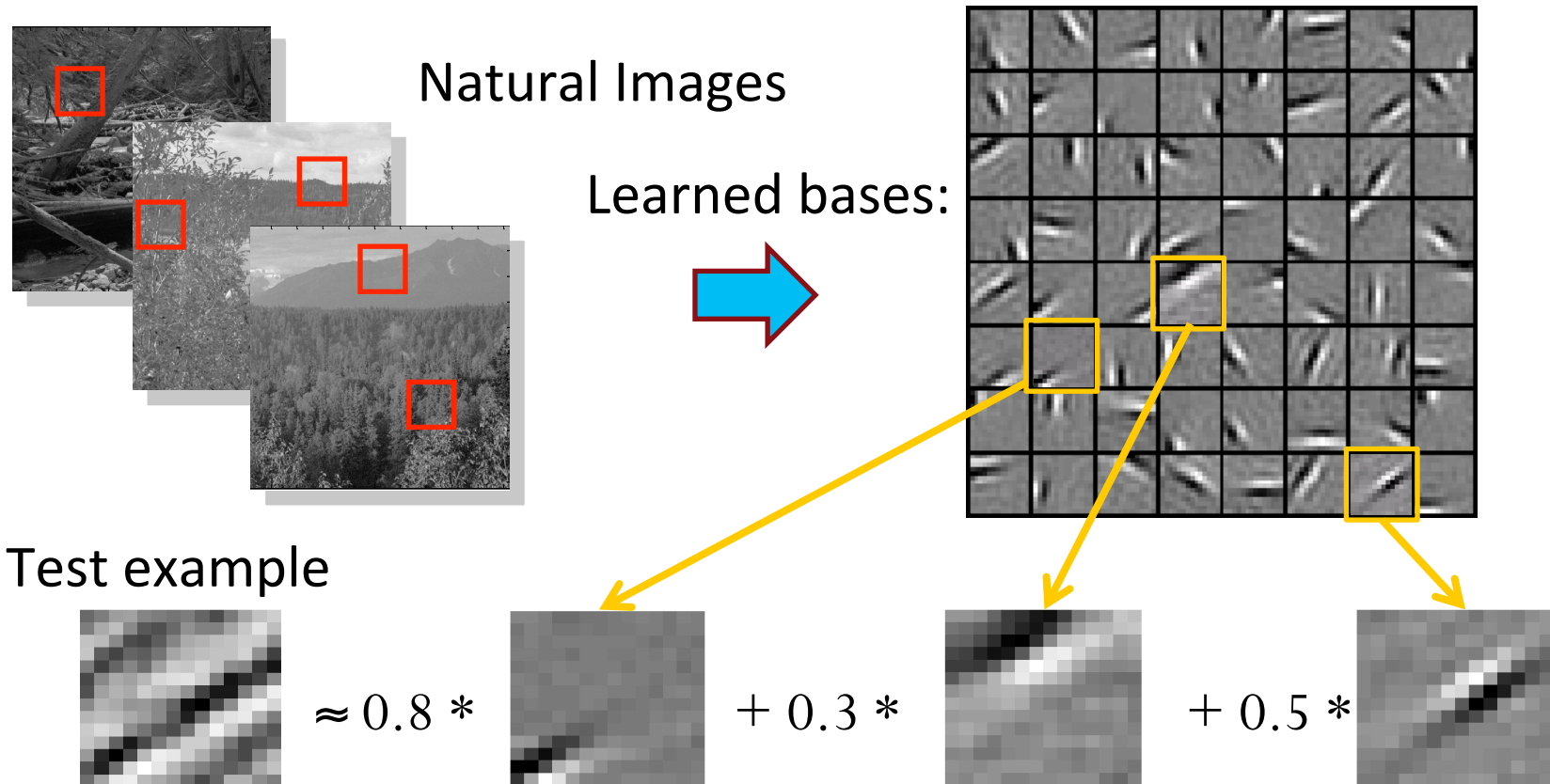
Directed Factor Models

- $P(h)$ factorizes into $P(h_1) P(h_2) \dots$
- Different priors:
 - PCA: $P(h_i)$ is Gaussian
 - ICA: $P(h_i)$ is non-parametric
 - **Sparse coding**: $P(h_i)$ is concentrated near 0
- Likelihood is typically Gaussian $x | h$ 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



$$h_i \quad x \quad = \quad .9 \quad x \quad + \quad .8 \quad x \quad + \quad .7 \quad x$$

Sparse autoencoder illustration for images

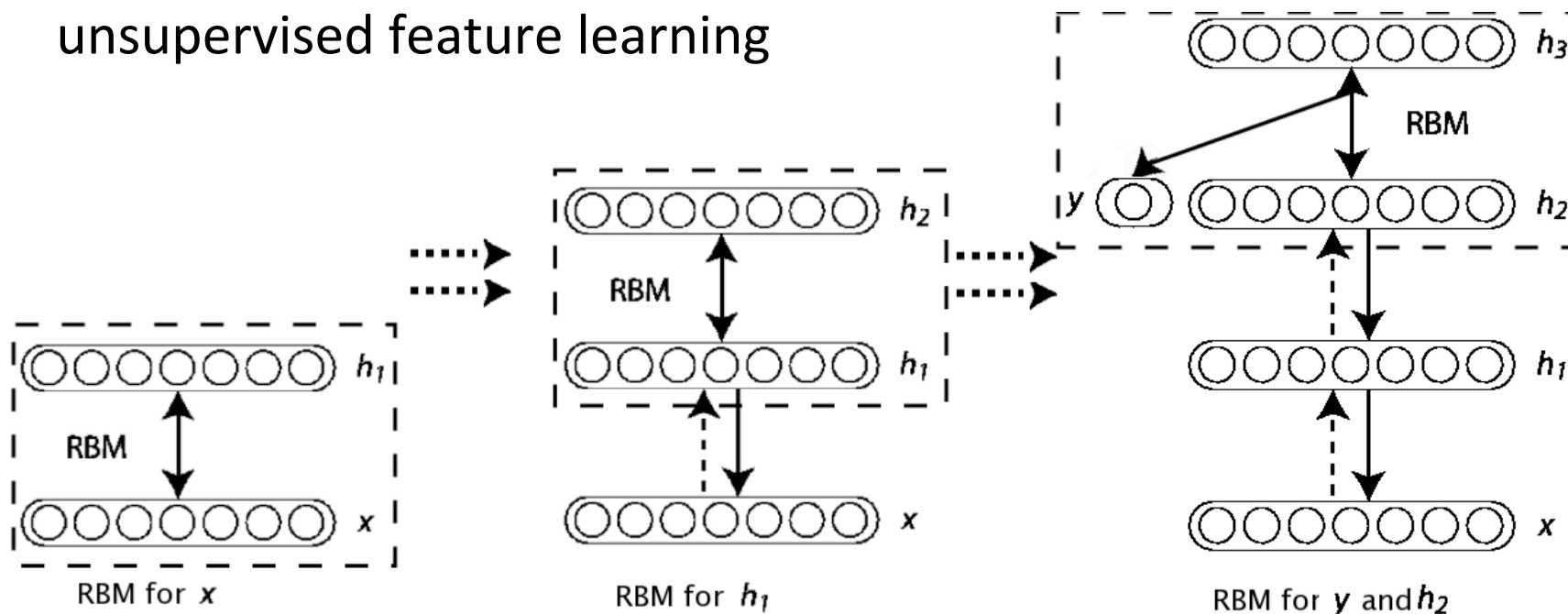


$$[a_1, \dots, a_{64}] = [0, 0, \dots, 0, \mathbf{0.8}, 0, \dots, 0, \mathbf{0.3}, 0, \dots, 0, \mathbf{0.5}, 0]$$

(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



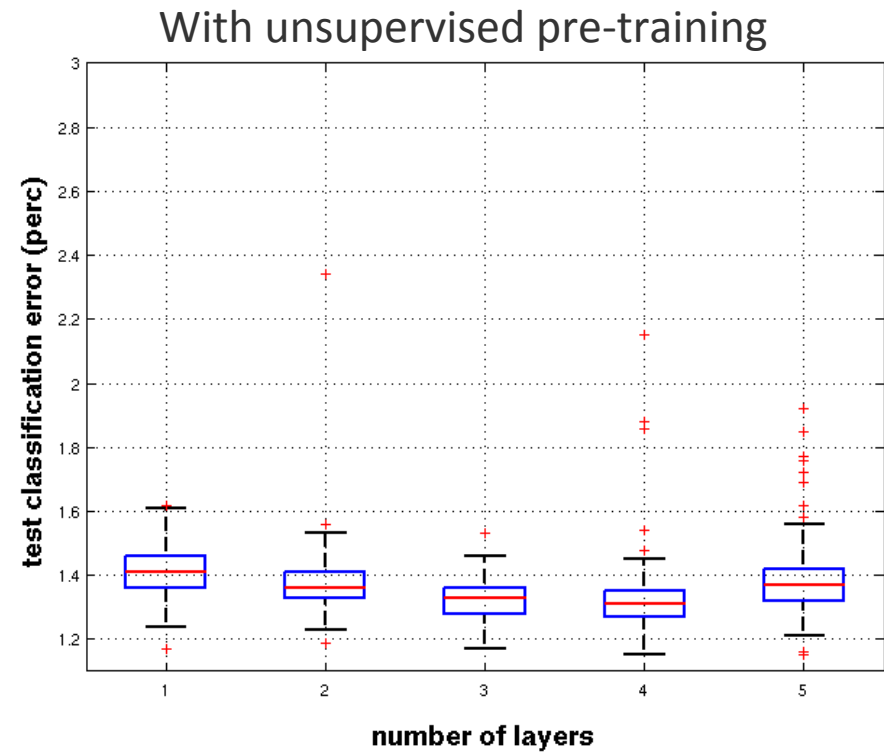
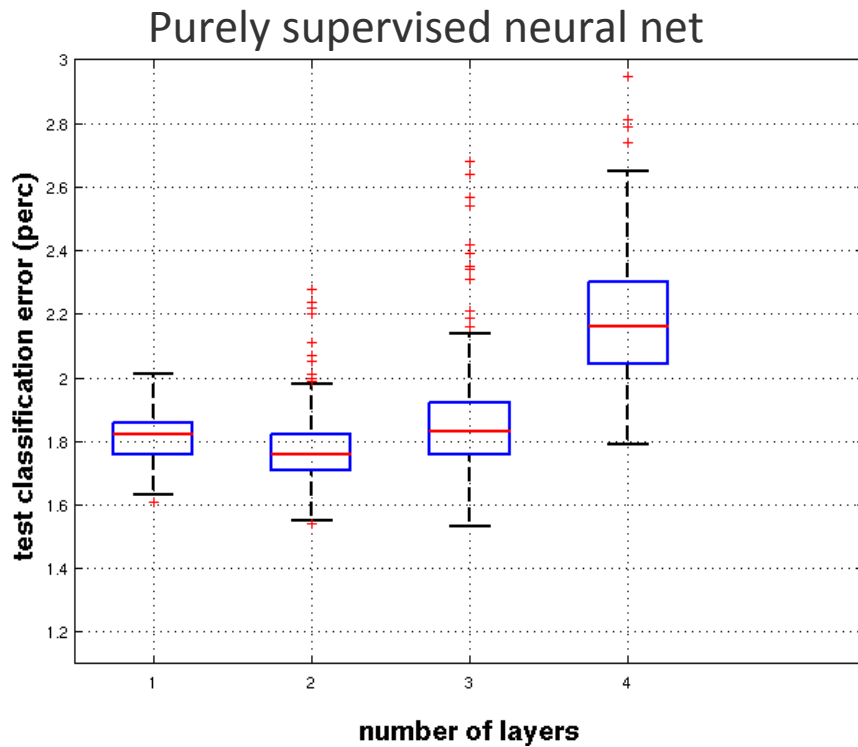
Stacking Restricted Boltzmann Machines (RBM) \rightarrow Deep Belief Network (DBN)

Effective deep Learning became possible through unsupervised pre-training

[Erhan et al., JMLR 2010]



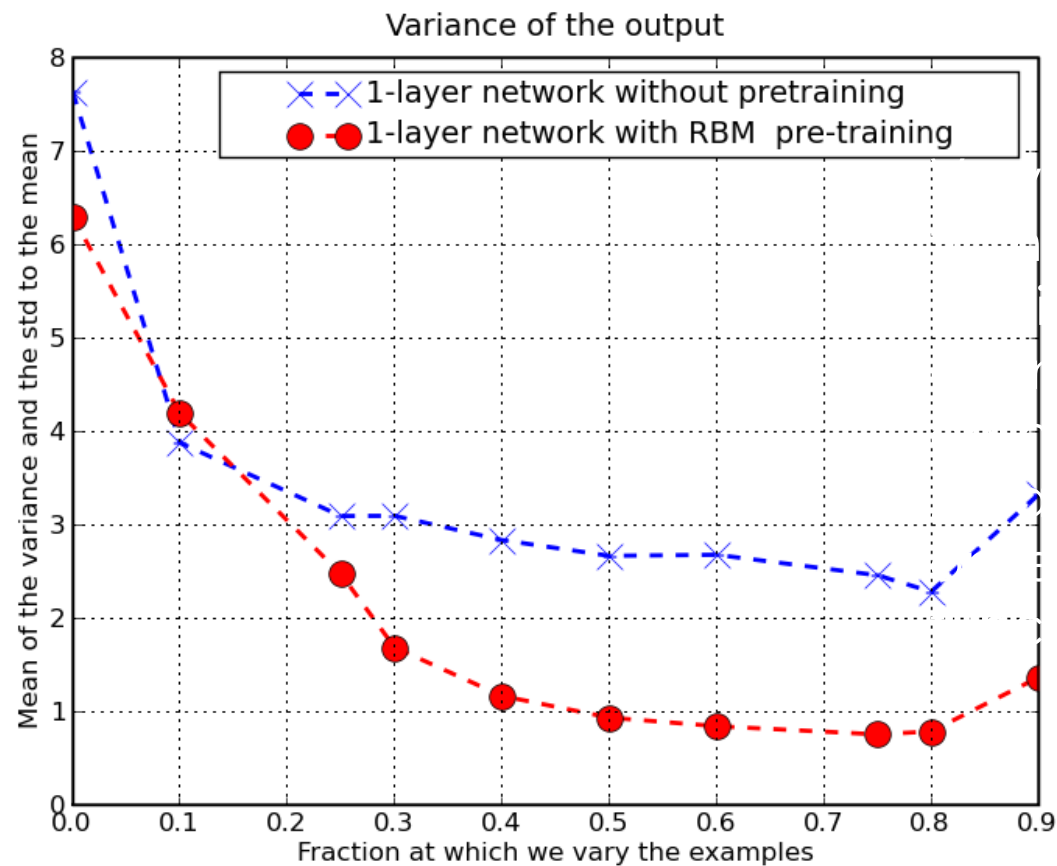
(with RBMs and Denoising Auto-Encoders)



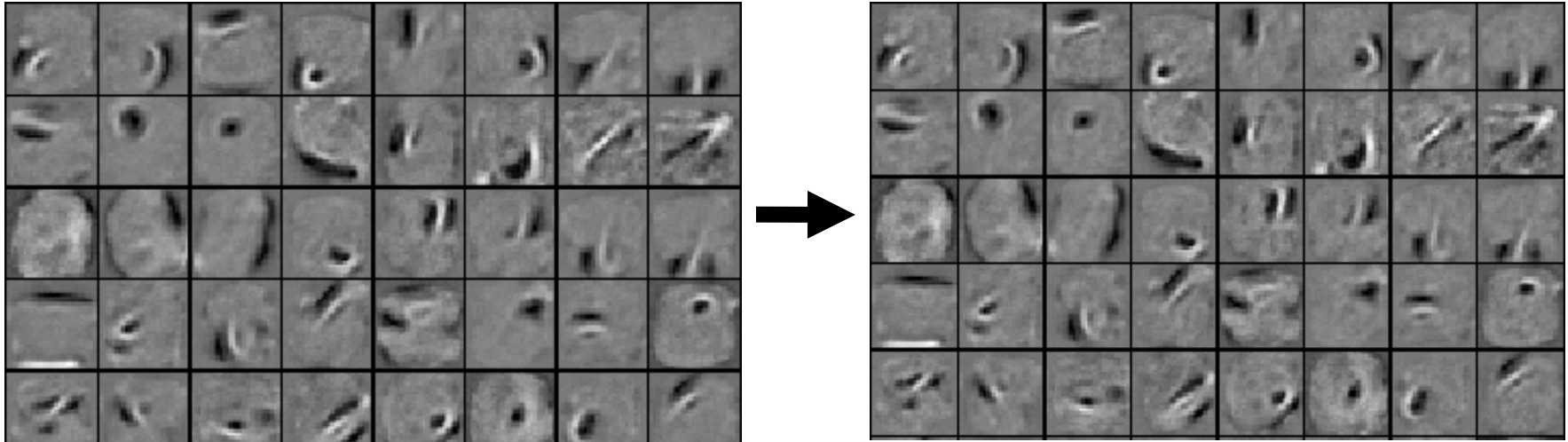
Optimizing Deep Non-Linear Composition of Functions Seems Hard

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

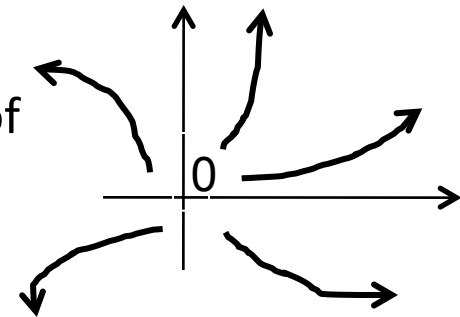
Initial Examples Matter More (critical period?)



Learning Dynamics of Deep Nets



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

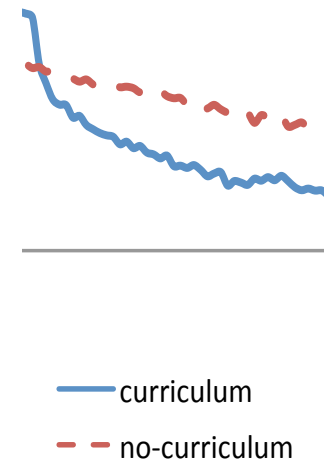


Order & Selection of Examples Matters

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



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



Understanding the difficulty of training deep feedforward neural networks

(Glorot & Bengio, AISTATS 2010)



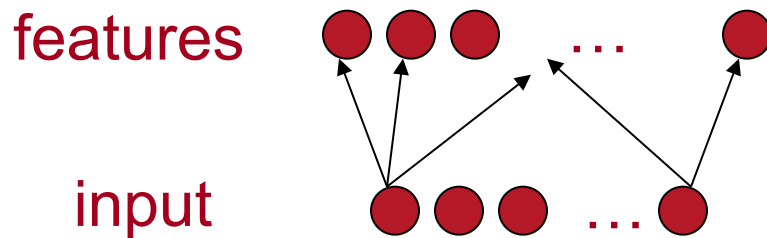
Study the activations and gradients

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

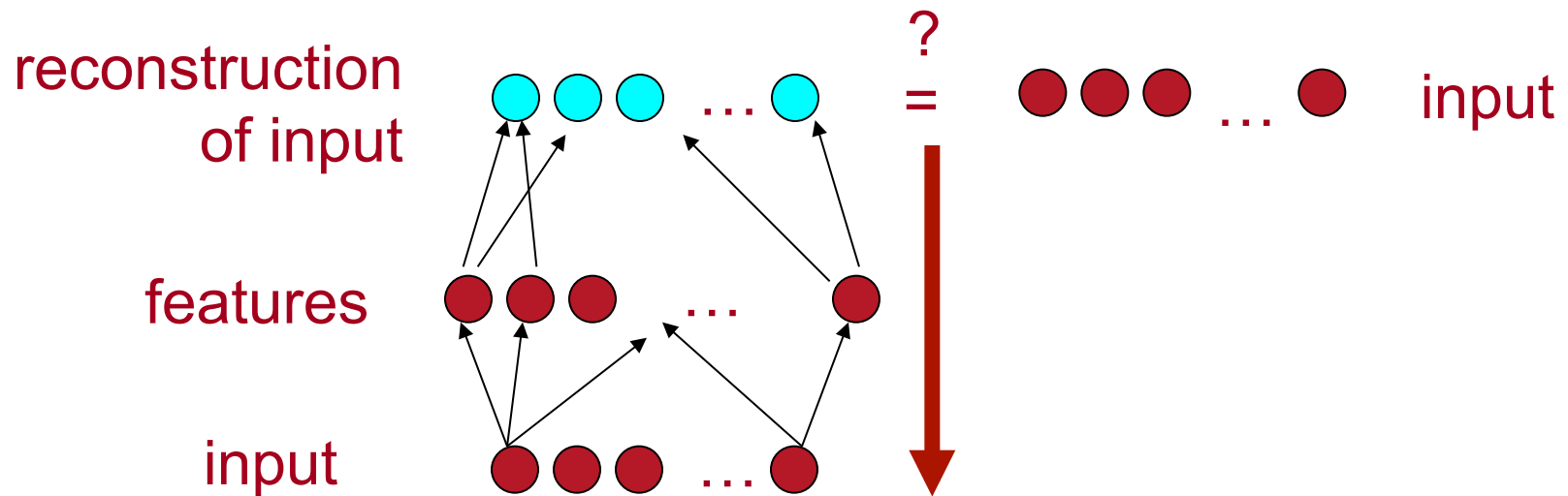
Layer-wise Unsupervised Learning

input ● ● ● ... ●

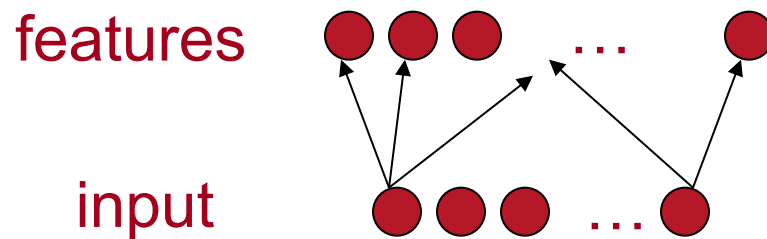
Layer-Wise Unsupervised Pre-training



Layer-Wise Unsupervised Pre-training



Layer-Wise Unsupervised Pre-training

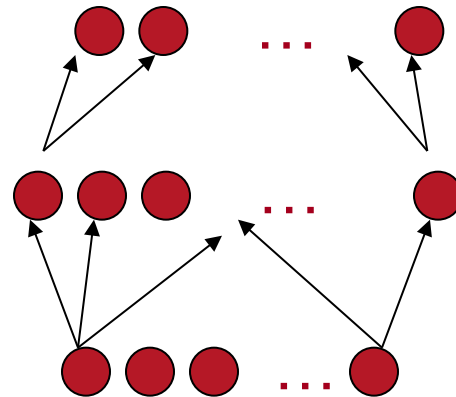


Layer-Wise Unsupervised Pre-training

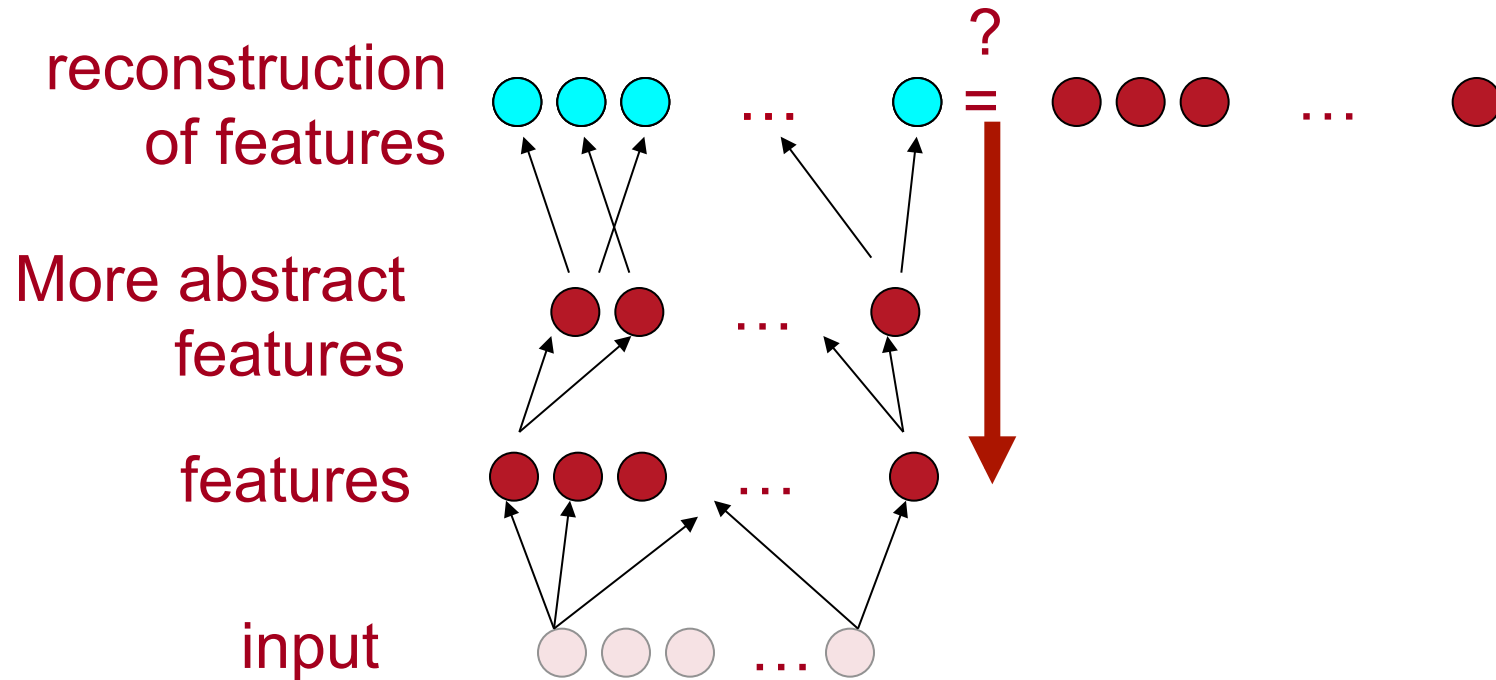
More abstract
features

features

input



Layer-wise Unsupervised Learning

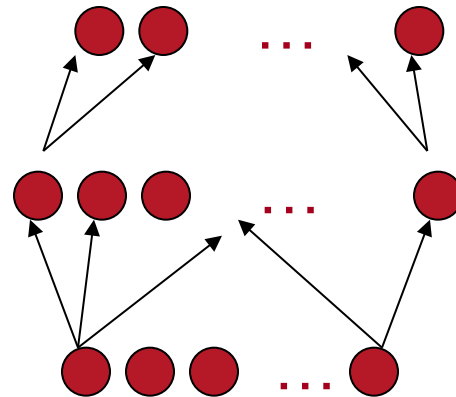


Layer-Wise Unsupervised Pre-training

More abstract
features

features

input



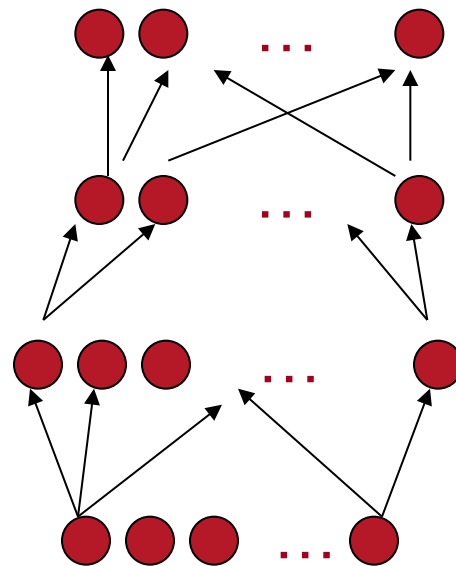
Layer-wise Unsupervised Learning

Even more abstract
features

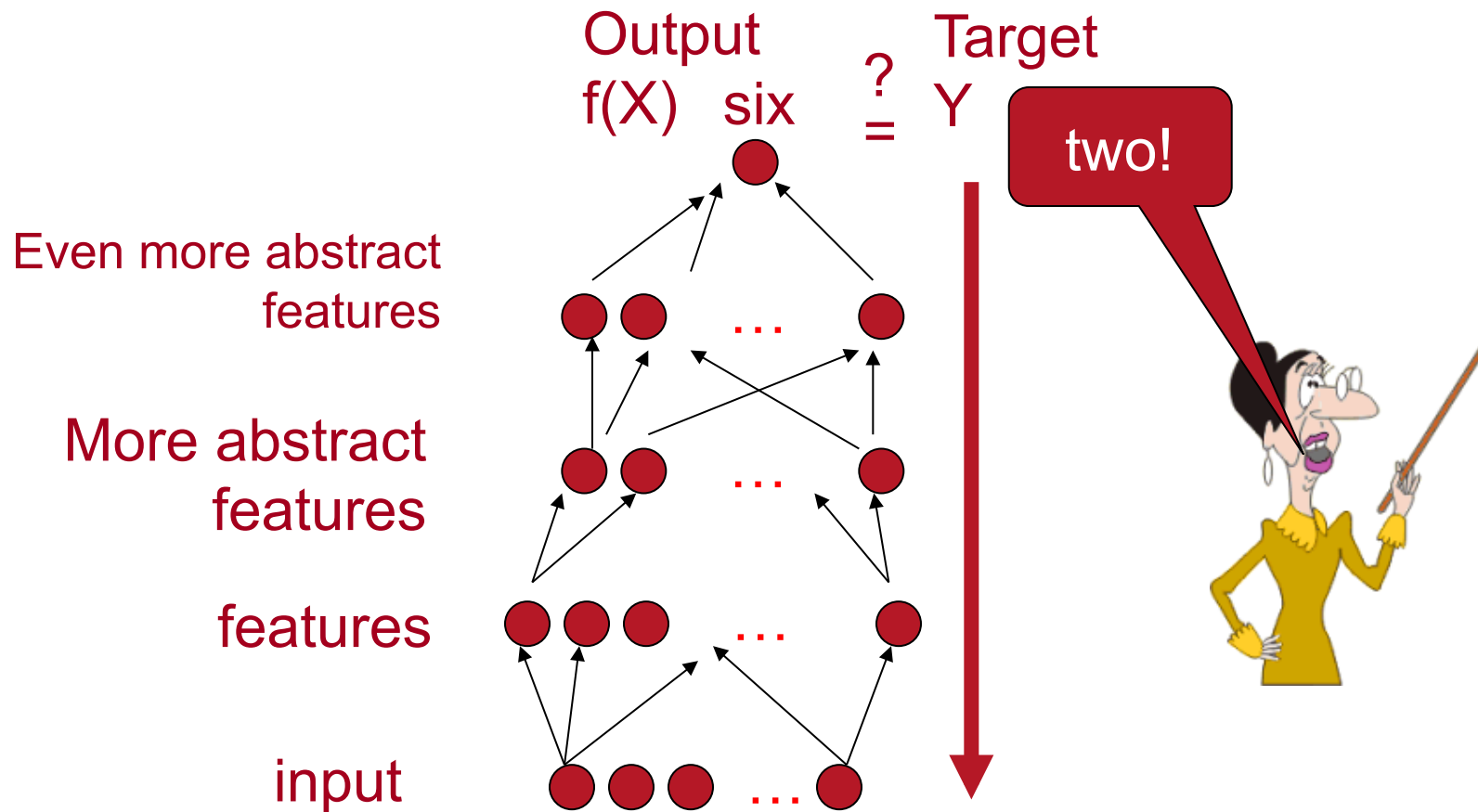
More abstract
features

features

input



Supervised Fine-Tuning



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

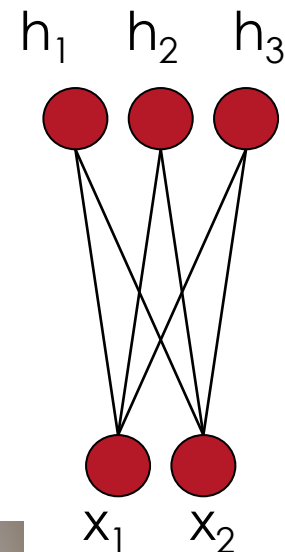
Restricted Boltzmann Machines

Undirected Models: the Restricted Boltzmann Machine

[Hinton et al 2006]



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



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

Boltzmann Machines & MRFs

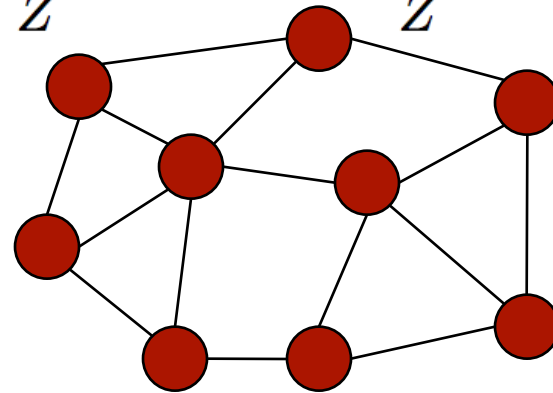
- Boltzmann machines:

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

- Markov Random Fields:

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

Soft constraint / probabilistic statement



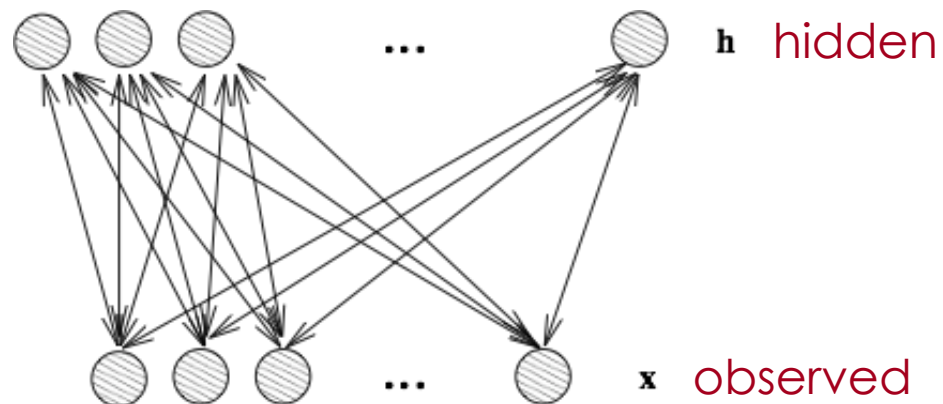
Undirected
graphical
models

- More interesting with latent variables!

Restricted Boltzmann Machine (RBM)

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

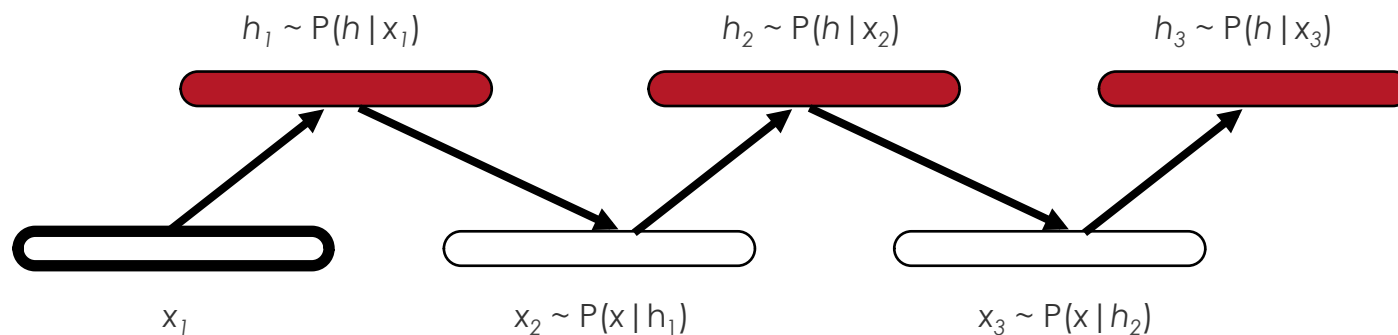
- A popular building block for deep architectures
- **Bipartite** undirected graphical model



Gibbs Sampling & Block Gibbs Sampling

- Want to sample from $P(X_1, X_2, \dots, X_n)$
 - **Gibbs sampling**
 - Iterate or randomly choose i in $\{1 \dots n\}$
 - Sample X_i from $P(X_i \mid X_1, X_2, \dots, X_{i-1}, X_{i+1}, \dots, X_n)$
- can only make small changes at a time! → **slow mixing**
- Note how fixed point samples from the joint.
- **Block Gibbs sampling**
 - X 's organized in blocks, e.g. $A=(X_1, X_2, X_3)$, $B=(X_4, X_5, X_6)$, $C=...$
 - Do Gibbs on $P(A, B, C, \dots)$, i.e.
 - Sample A from $P(A \mid B, C)$
 - Sample B from $P(B \mid A, C)$
 - Sample C from $P(C \mid A, B)$, and iterate...
 - Larger changes → **faster mixing**

Gibbs Sampling in RBMs



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

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

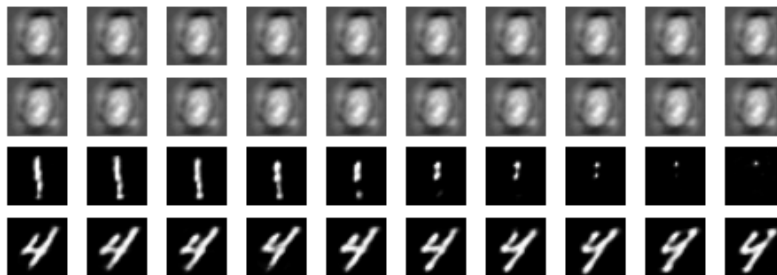
- Easy inference
- Efficient **block Gibbs** sampling $x \rightarrow h \rightarrow x \rightarrow h \dots$

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

Problems with Gibbs Sampling

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

RBM trained by CD on MNIST



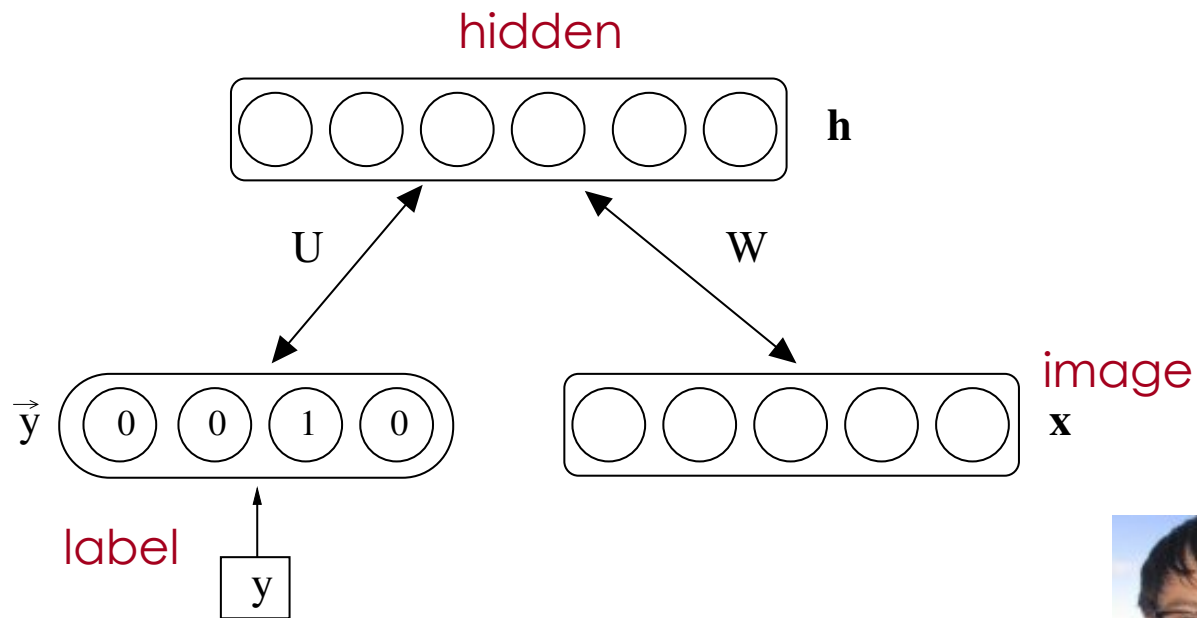
Chains from **random state**

Chains from **real digits**



(Desjardins et al 2010)

RBM with (image, Label) visible units



(Larochelle & Bengio 2008)

RBMs are Universal Approximators

(Le Roux & Bengio 2008)



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

RBM Conditionals Factorize

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

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

Factorization of the Free Energy

Let the energy have the following general form:

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

Then

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

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

Energy-Based Models Gradient

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

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

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

Boltzmann Machine Gradient

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

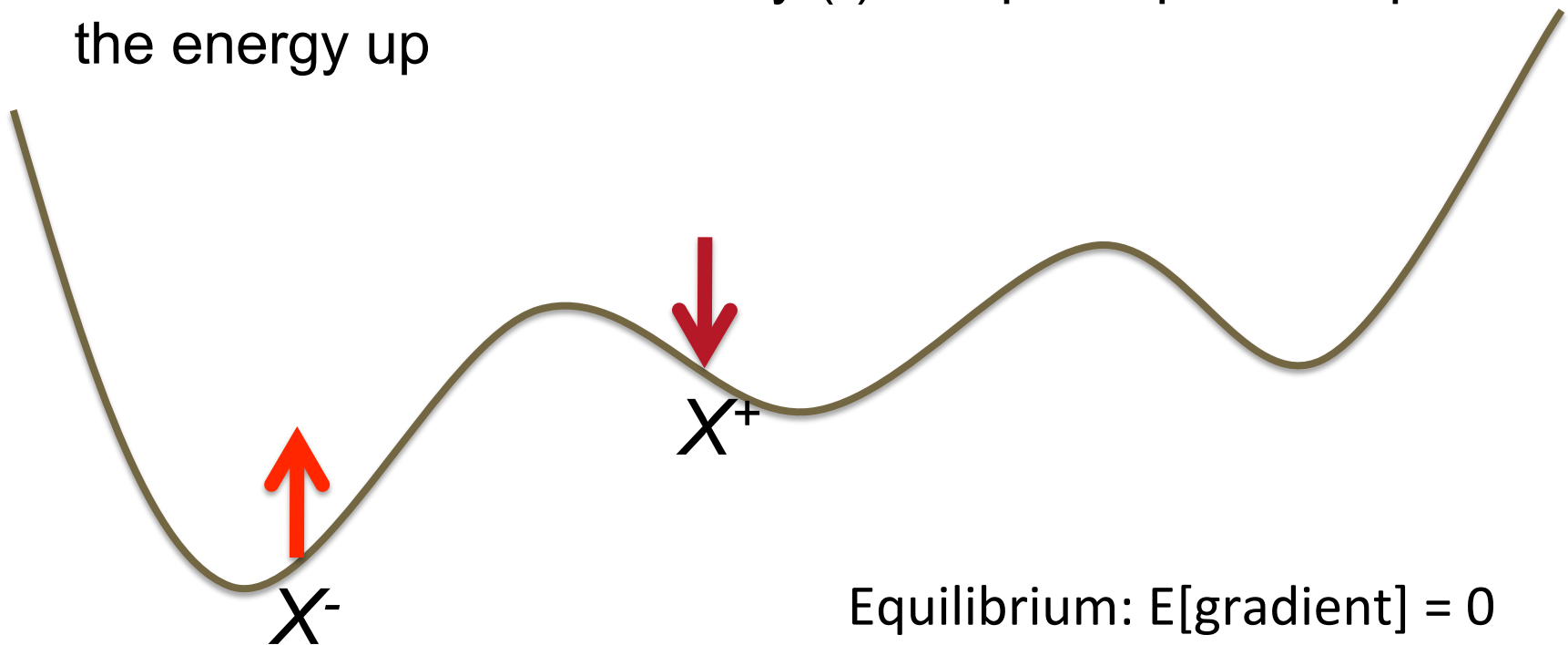
- Gradient has two components:

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

- 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

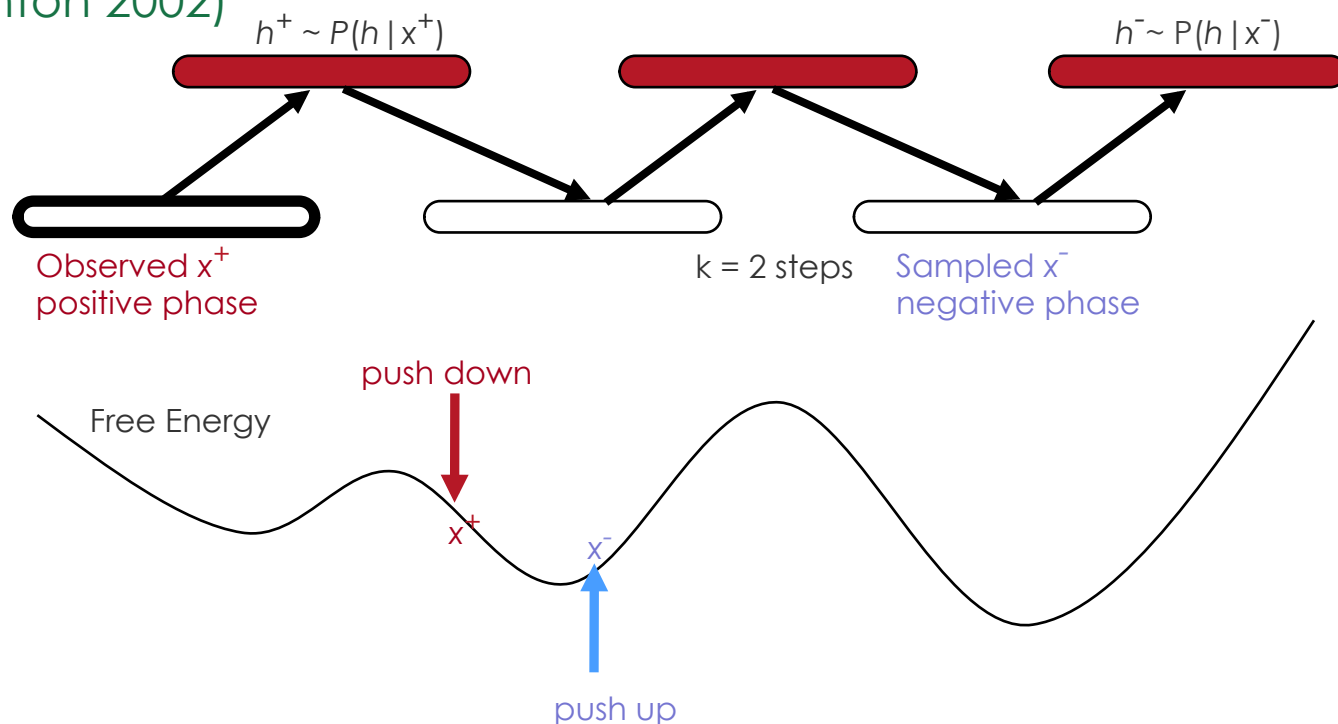
Tempered MCMC: use higher temperature to escape modes

Contrastive Divergence



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

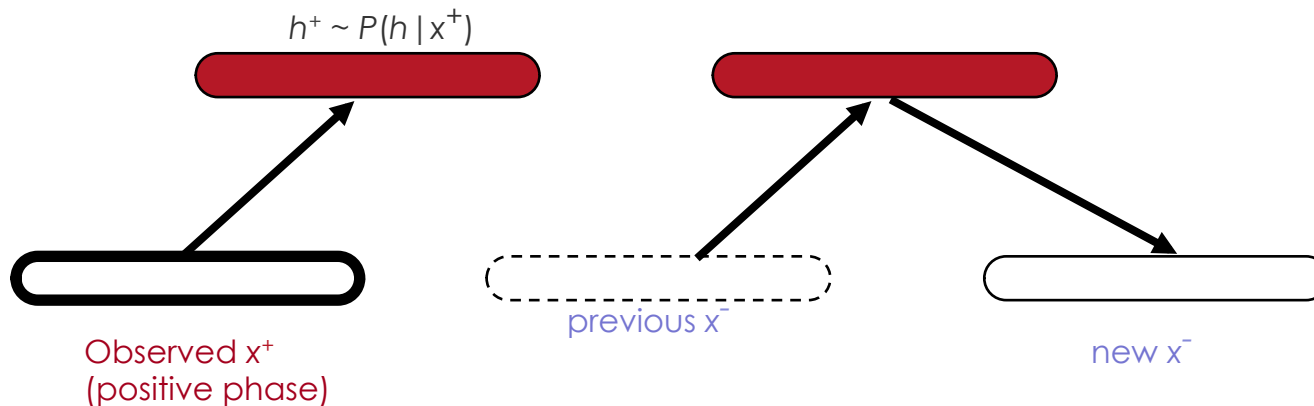
(Hinton 2002)



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

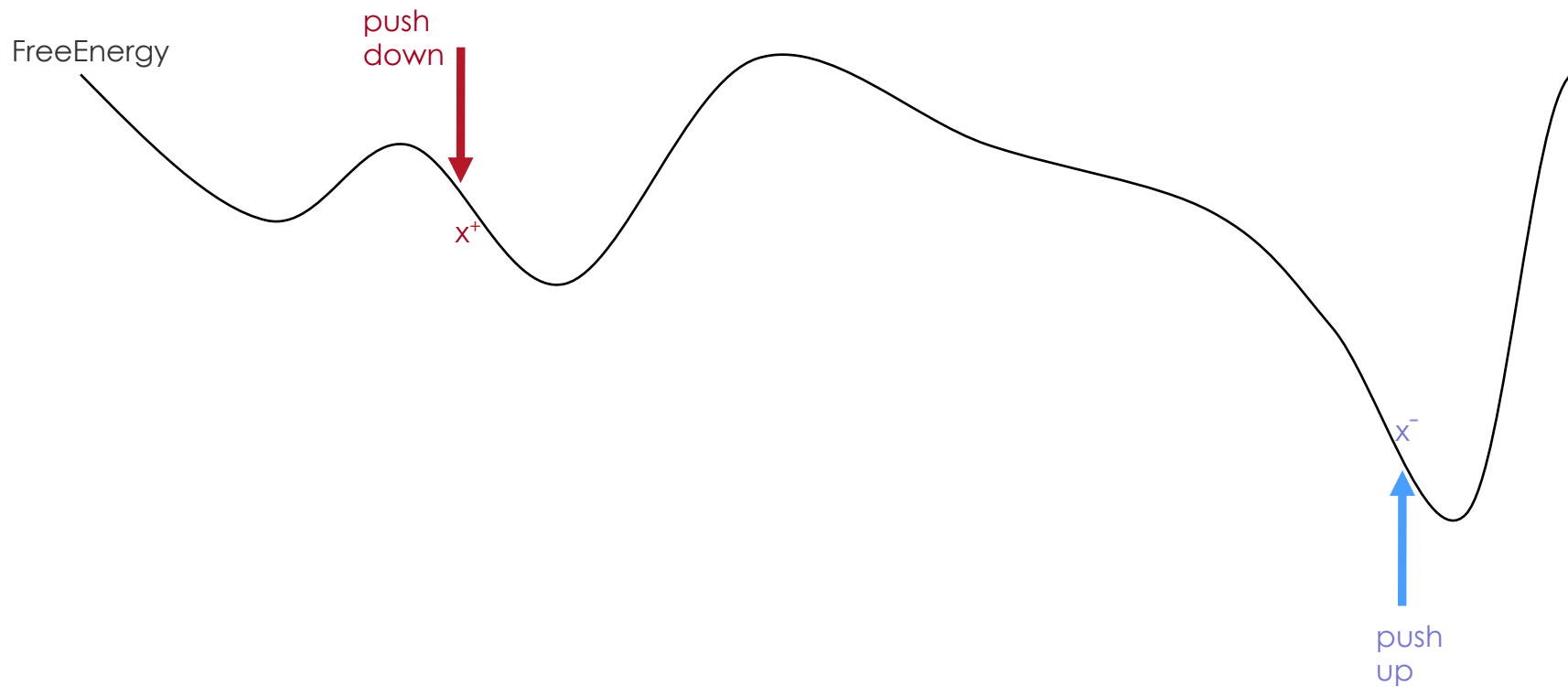
Run negative Gibbs chain in background while weights slowly change (Younes 1999, Tieleman 2008):

- 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



PCD/SML + Large Learning rate

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

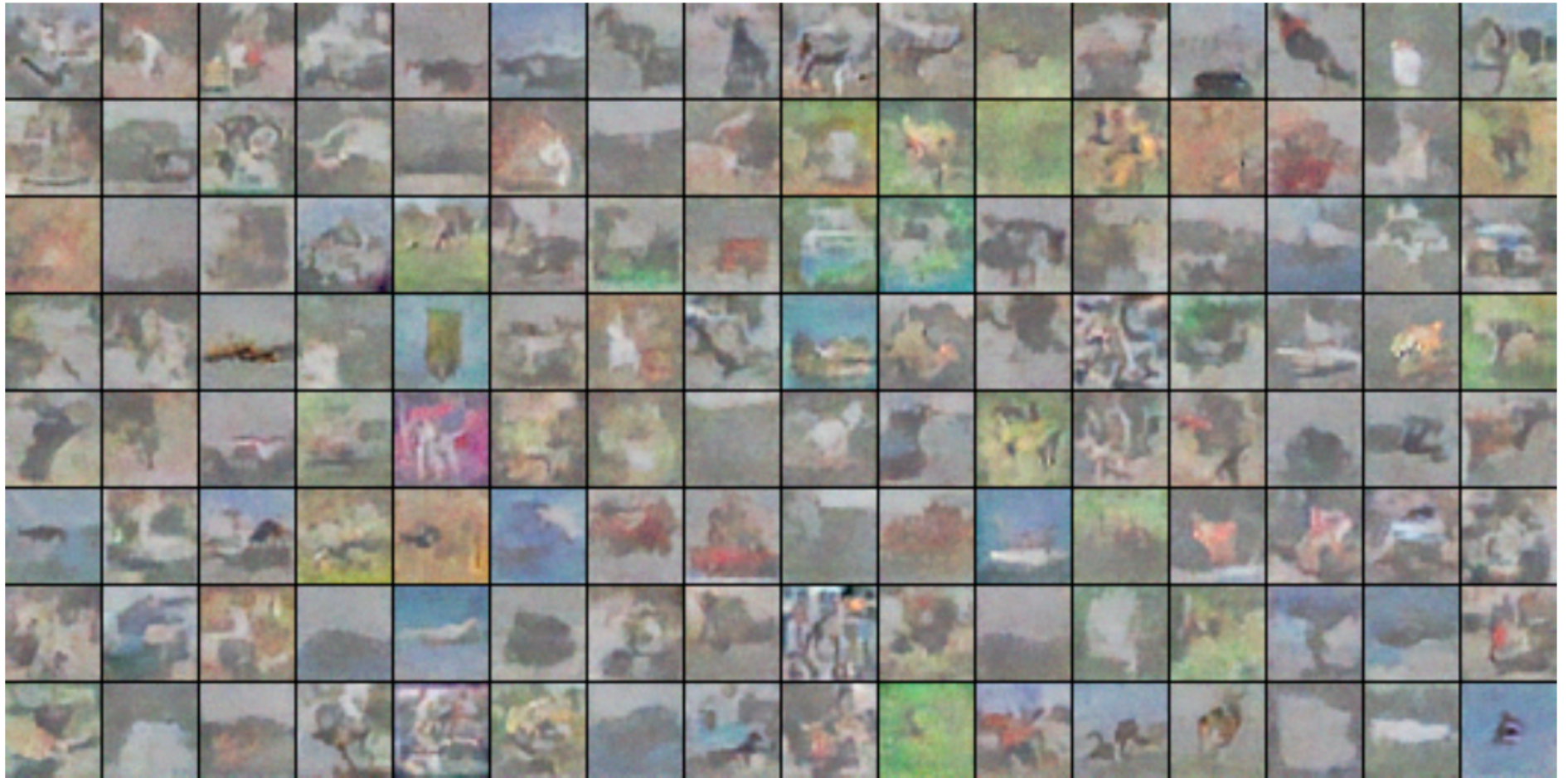


Some RBM Variants

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



Convolutionally Trained Spike & Slab RBMs Samples

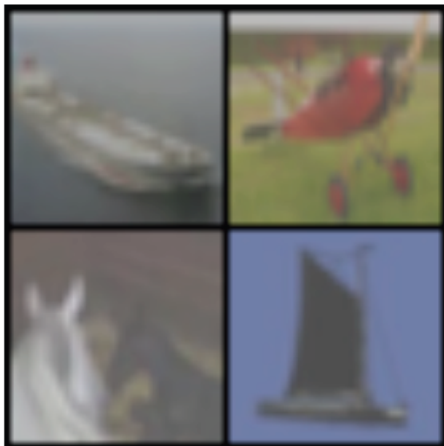


ssRBM is not Cheating

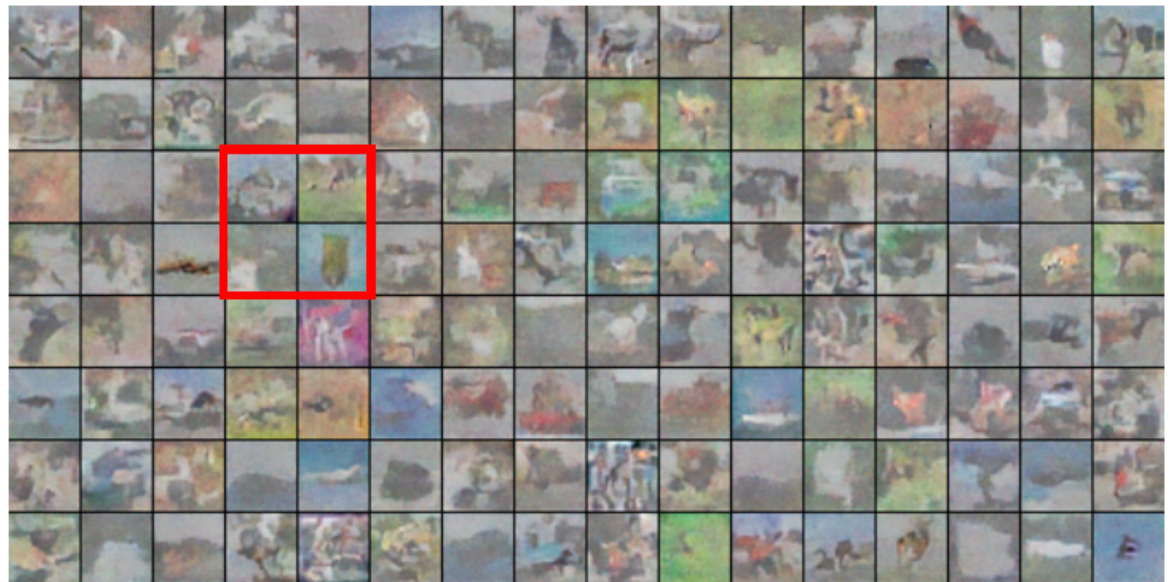
Samples from μ -ssRBM:



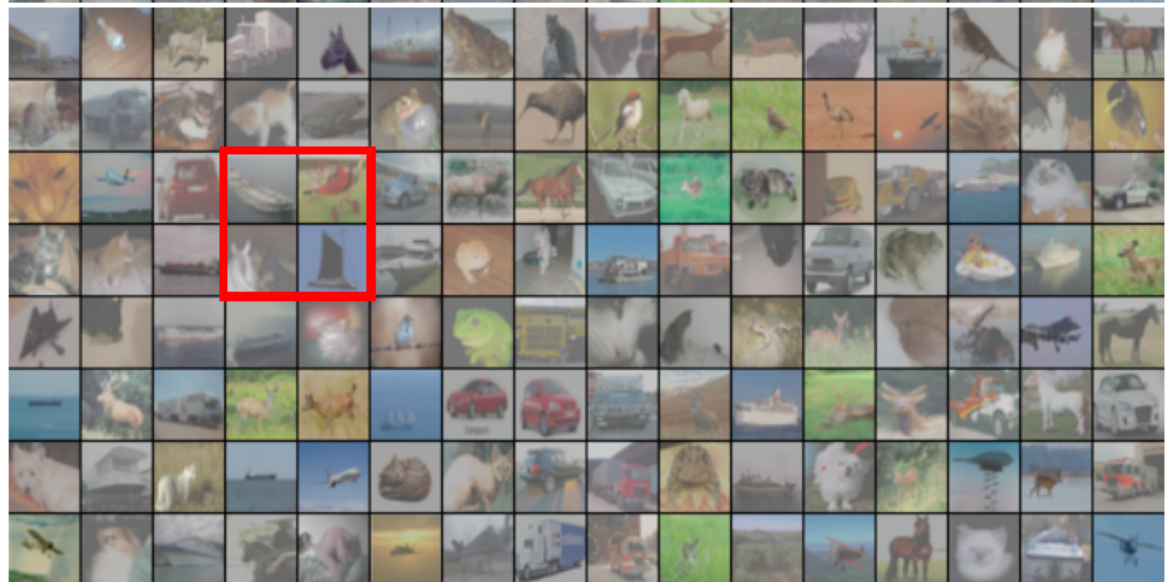
Nearest examples in CIFAR:
(least square dist.)



Generated samples



Training examples



Spike & Slab RBMs

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

Model conditional covariance of pixels (given hidden units)

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

Hidden representation decomposed into a product s^*h , h is binary, s is real
 s^*h is often 0 (naturally sparse)

Spike & Slab RBMs

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

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

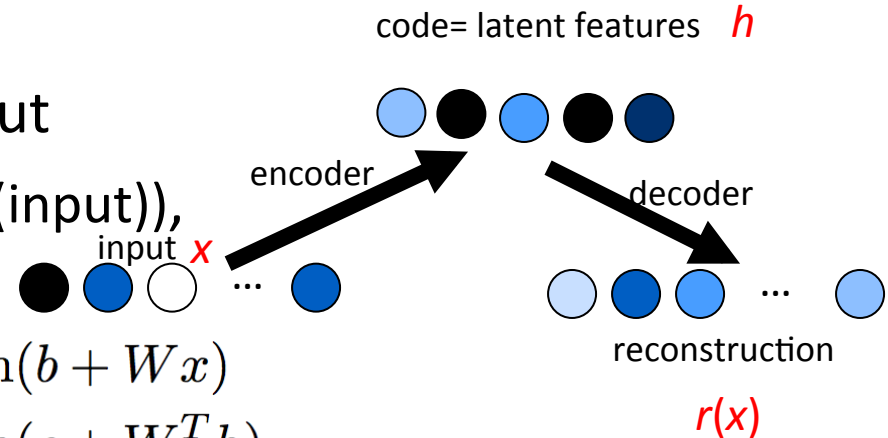
$$p(v | s, h) = \mathcal{N}\left(C_{v|s,h} \sum_{i=1}^N W_i s_i h_i, C_{v|s,h}\right)$$

Can use efficient 3-way Gibbs sampling

Auto-Encoders & Variants

Auto-Encoders

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



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

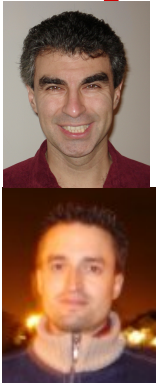
$$\text{reconstruction} = \tanh(c + W^T h)$$

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

- 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

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

Traditional Directed $X|\theta$ Models

$$P(X, \theta) = P(X|\theta)P(\theta)$$

$$P(X|\theta) = \frac{e^{-E_\theta(X)}}{Z_\theta}$$

$$\frac{\partial \log Z_\theta}{\partial \theta} = - \sum_X P(X|\theta) \frac{\partial E_\theta(X)}{\partial \theta}$$

What are regularized auto-encoders learning exactly?

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

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

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

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

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

Joint Parameter-Data Energy (JEPADA)

- Getting rid of the partition function problem
- Sampling X given θ , even when previously there was no probabilistic interpretation to $E(X, \theta)$
- Sampling θ given X (Bayesian)
- Inference and decision based on the model for which θ was really tuned.

- BUT WHAT MATHEMATICAL FORMS MAKE SENSE?
Reconstruction error and pseudo-likelihood-like things seem to work well. What else?

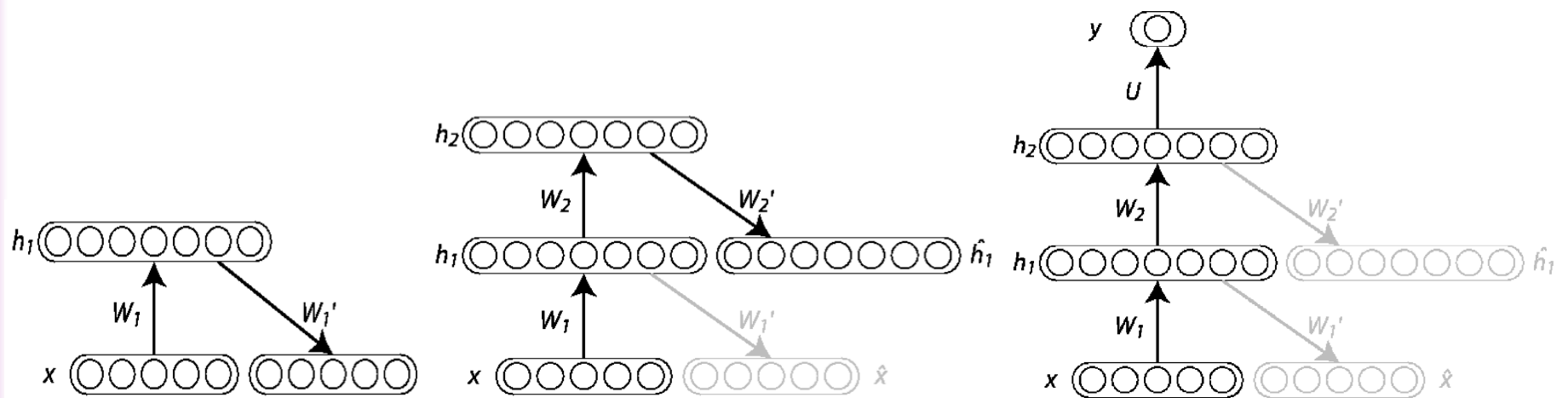
I think I finally understand what auto-encoders do!

- Try to carve holes in $\|r(x)-x\|^2$ at training examples

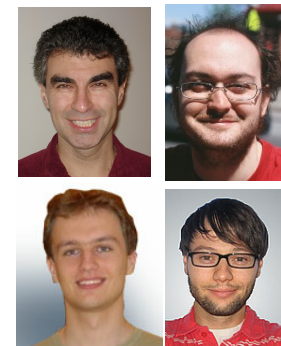


- Vector $r(x)-x$ points in direction of increasing prob., i.e. estimate score = $d \log p(x) / dx$: learn **score** vector field = **local mean**
- Generalize (*valleys*) in between above holes to form *manifolds*
 - $dr(x)/dx$ estimates the **local covariance** and is linked to the Hessian $d^2 \log p(x) / dx^2$
- Regularized AEs estimate 1st and 2nd local moments of the density (imagine a ball around each x), which allows to **sample**

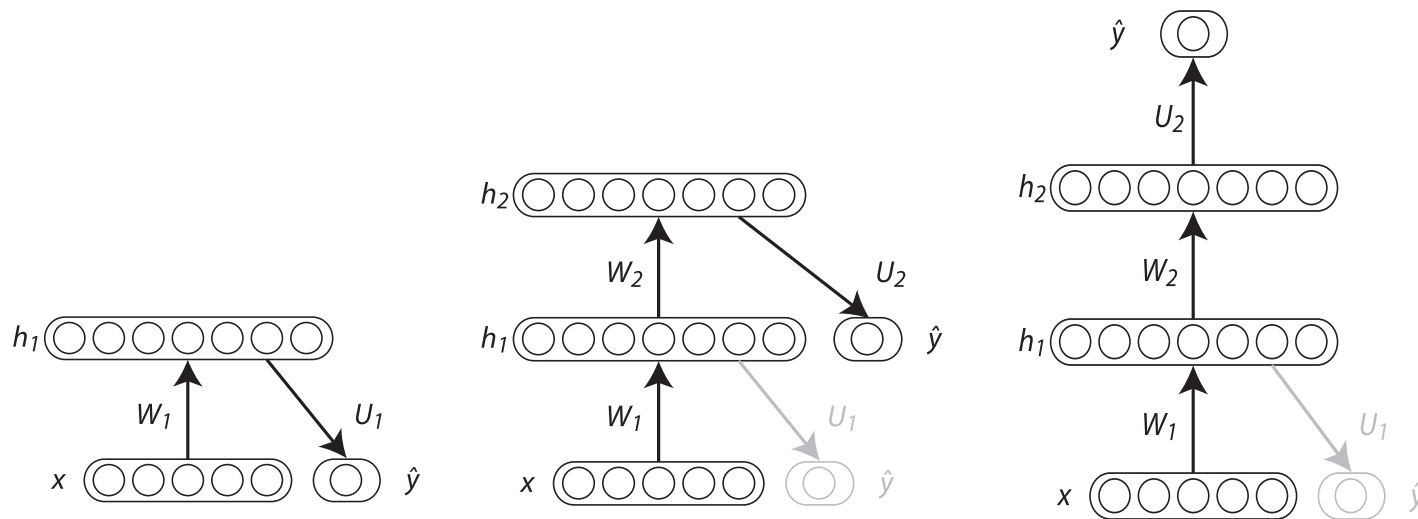
Stacking Auto-Encoders



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



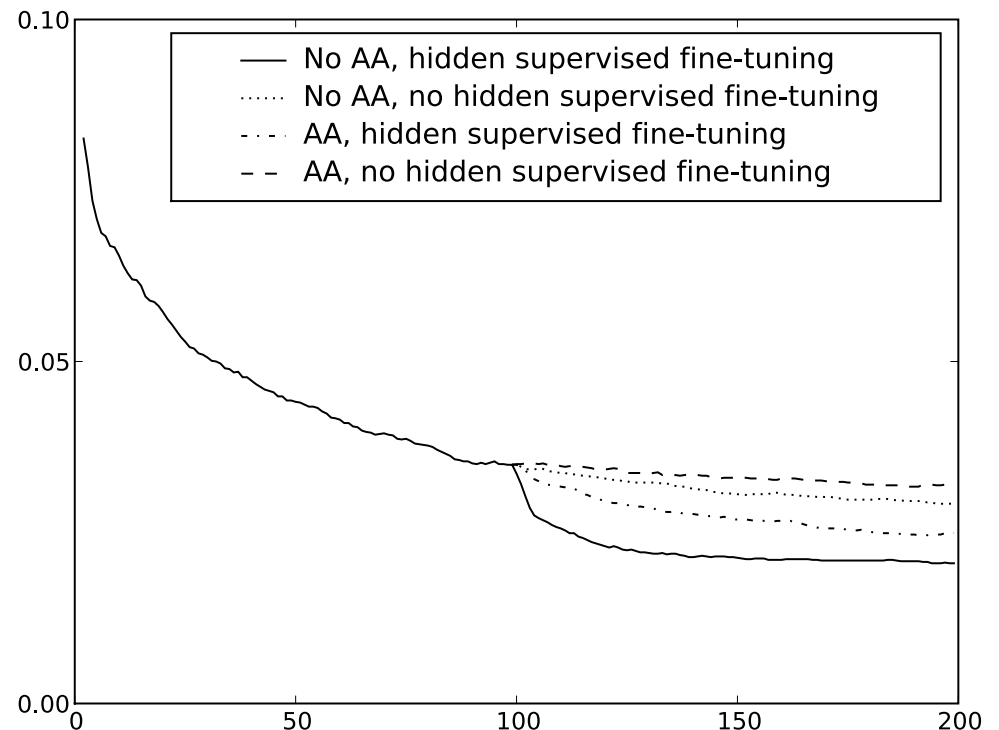
Greedy Layerwise Supervised Training



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

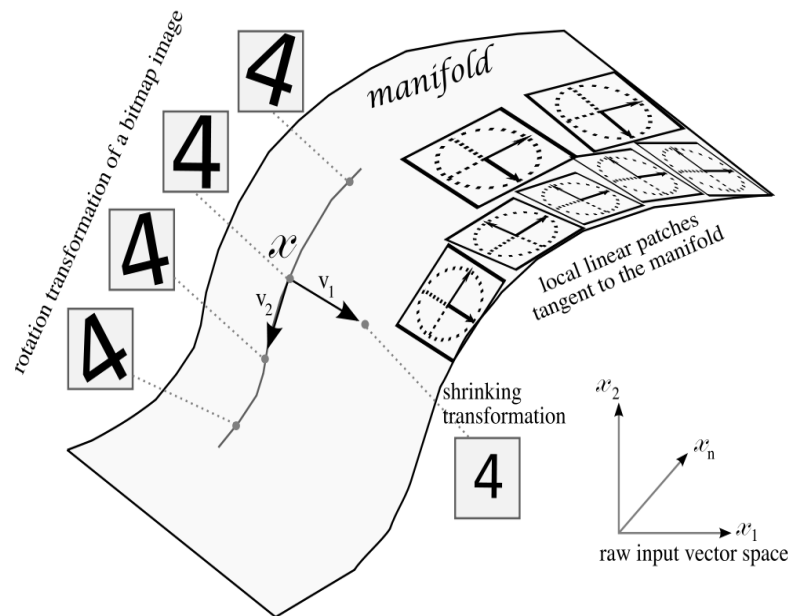


(Auto-Encoder) Reconstruction Loss

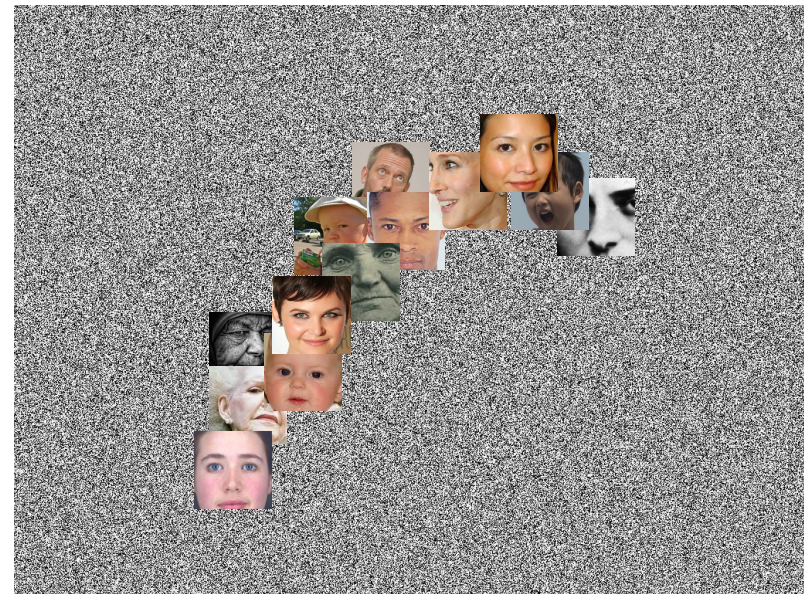
- Discrete inputs: cross-entropy for binary inputs
 - $-\sum_i x_i \log r_i(x) + (1-x_i) \log(1-r_i(x))$ (with $0 < r_i(x) < 1$)or log-likelihood reconstruction criterion, e.g., for a multinomial (one-hot) input
 - $-\sum_i x_i \log r_i(x)$ (where $\sum_i r_i(x) = 1$, summing over subset of inputs associated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

Manifold Learning

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

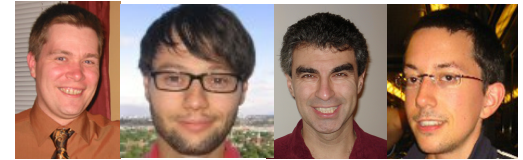


- variable dimension locally?
- Soft # of dimensions?

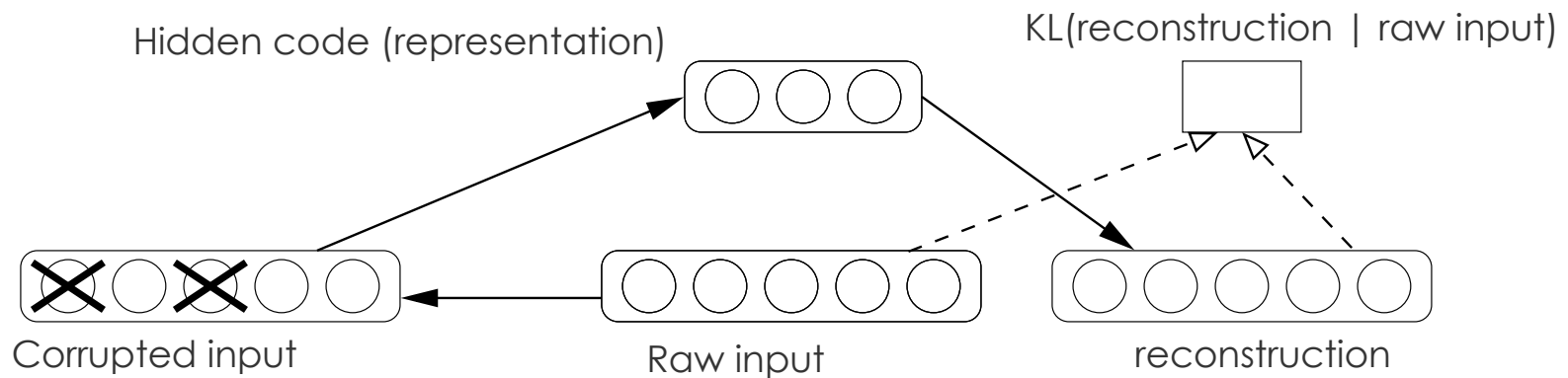


Denoising Auto-Encoder

(Vincent et al 2008)



- Corrupt the input
- Reconstruct the uncorrupted input



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

Denoising Auto-Encoder

- Learns a vector field pointing towards higher probability direction

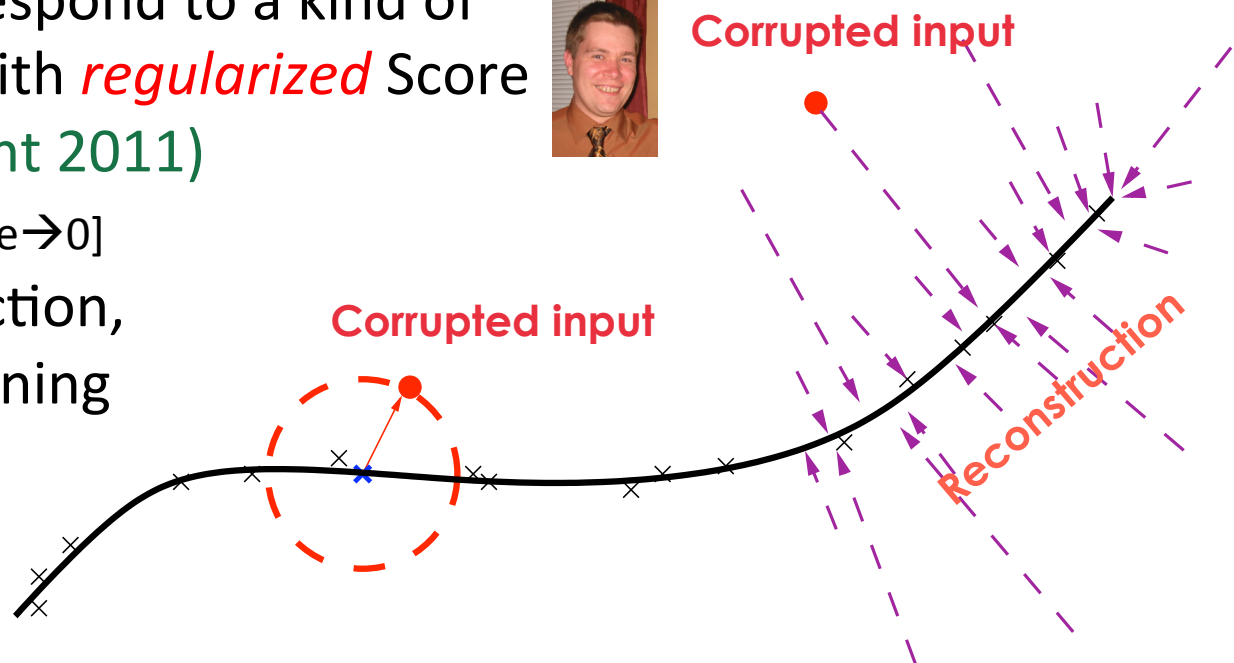
$$r(x)-x \approx d \log p(x) / dx$$

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

[equivalent when noise $\rightarrow 0$]

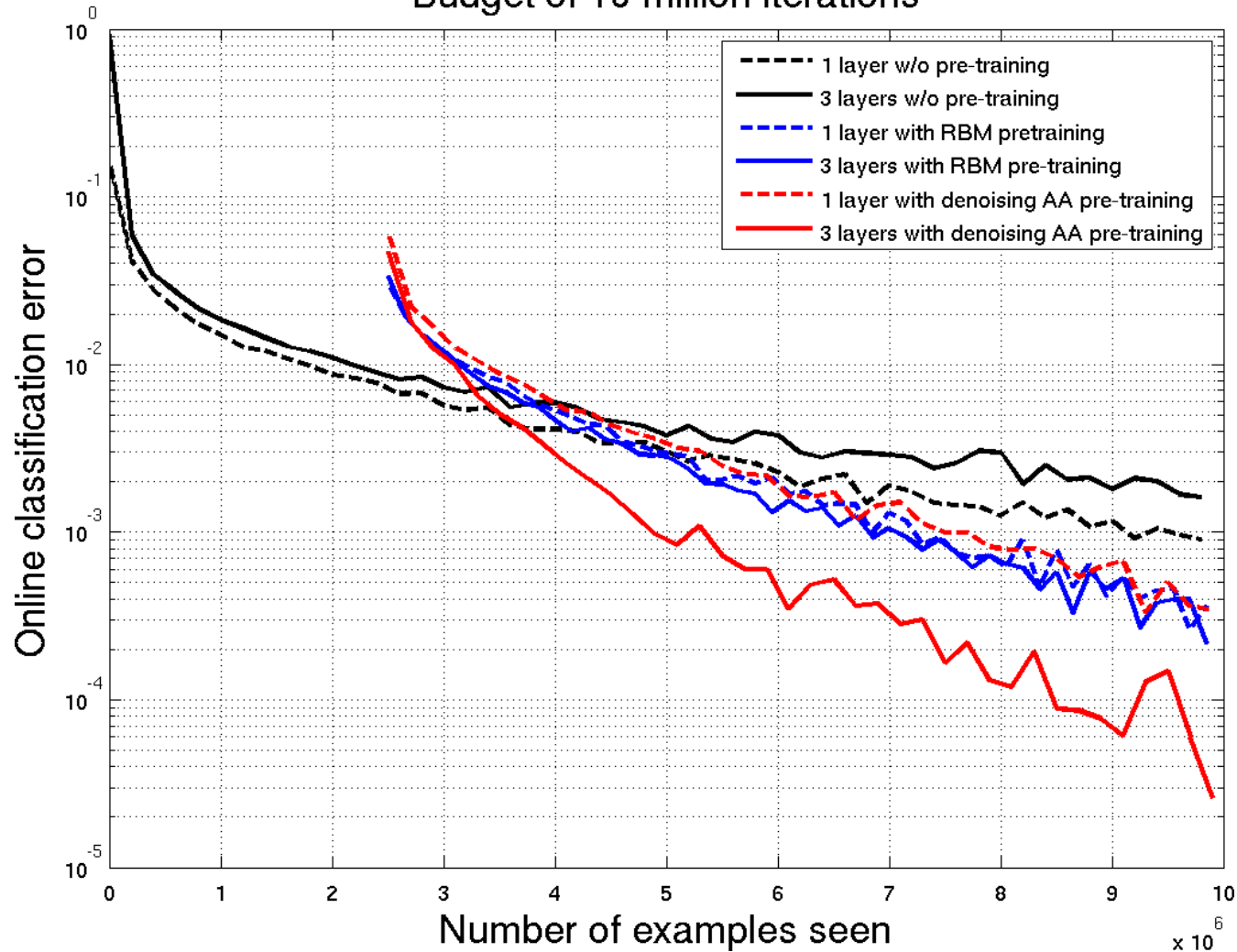
- No partition function, can measure training criterion

prior: examples concentrate near a lower dimensional "manifold"



Stacked Denoising Auto-Encoders

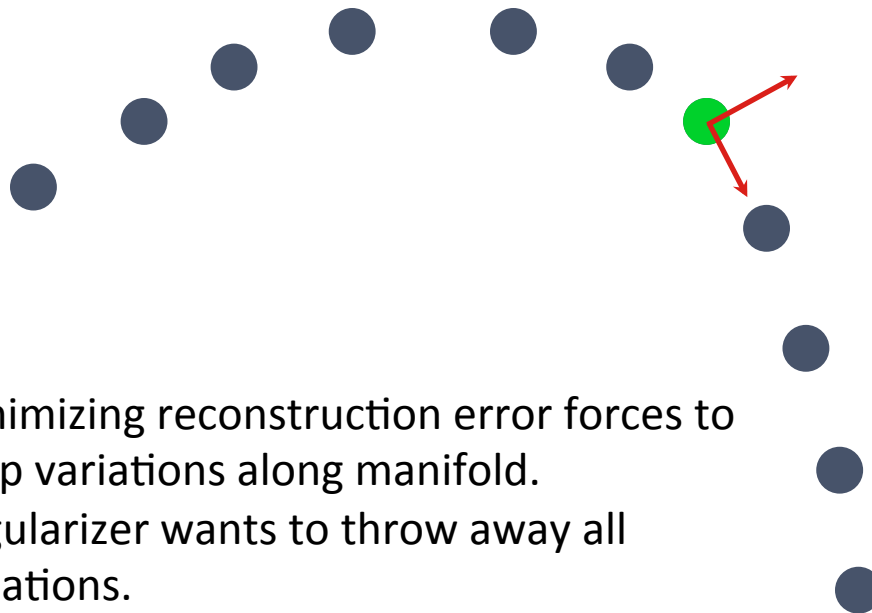
Budget of 10 million iterations



Infinite MNIST

Note how advantage of better initialization does not vanish like other regularizers as #examples $\rightarrow \infty$

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



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

Contractive Auto-Encoders



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

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

Training criterion:

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

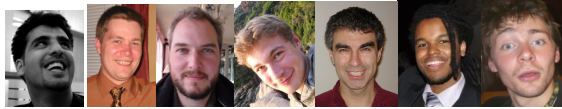
wants contraction in all directions

cannot afford contraction in manifold directions

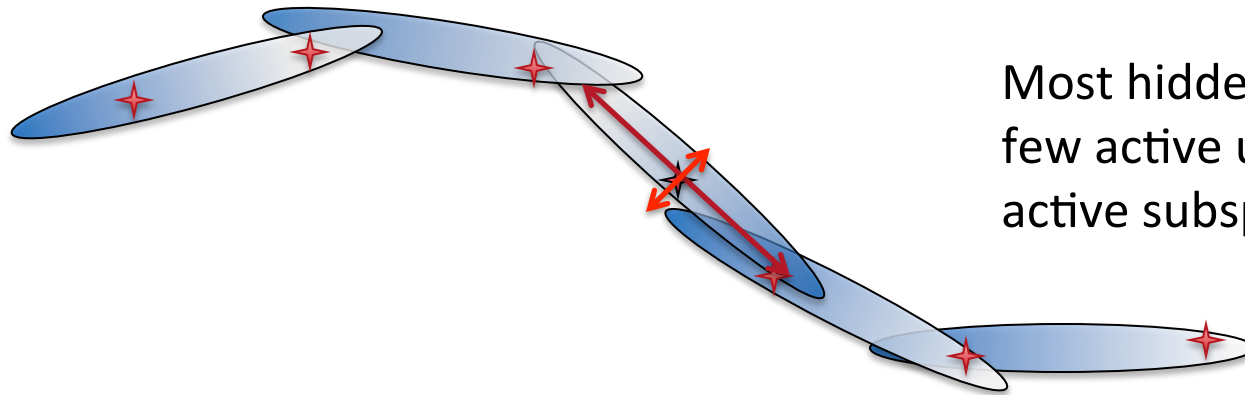
If $h_j = \text{sigmoid}(b_j + W_j x)$

$$(\frac{dh_j(x)}{dx_i})^2 = h_j^2(1-h_j)^2 W_{ji}^2$$

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)



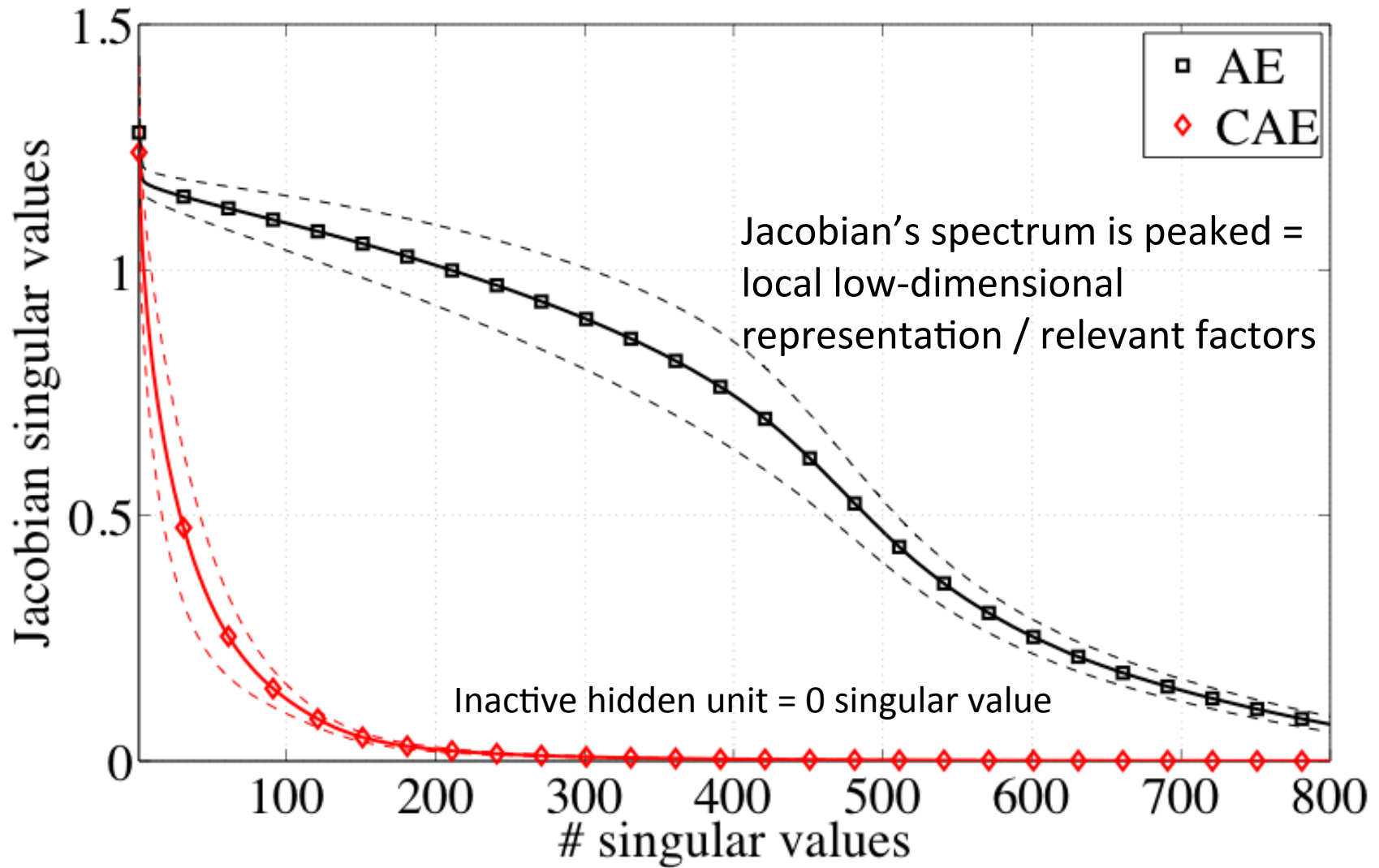
Most hidden units saturate:
few active units represent the
active subspace (local chart)

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

CIFAR-10



Contractive Auto-Encoders

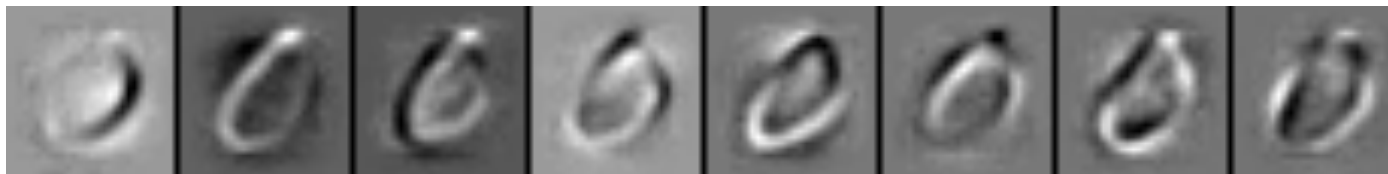
Benchmark of medium-size datasets on which several deep learning algorithms had been evaluated (Larochelle et al ICML 2007)

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

Input Point



Tangents



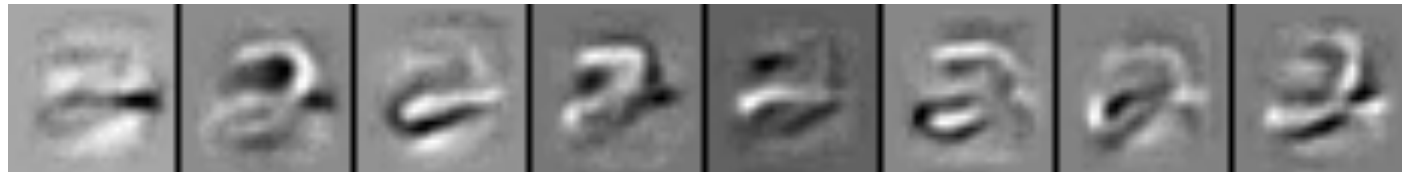
$$\text{Input Point} + 0.5 \times \text{Tangent} = \text{Result}$$

MNIST

Input Point



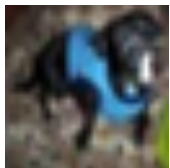
Tangents



MNIST Tangents

Distributed vs Local (CIFAR-10 unsupervised)

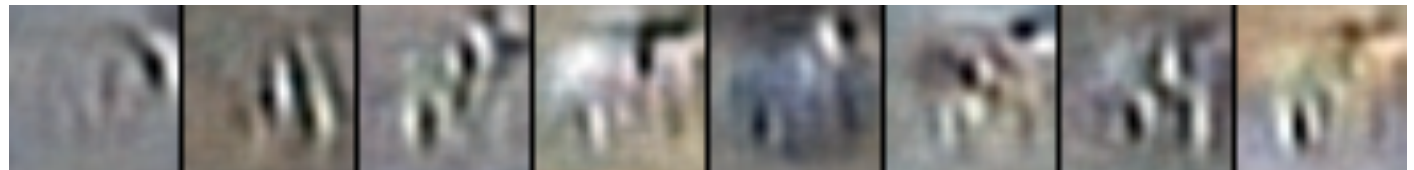
Input Point



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

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

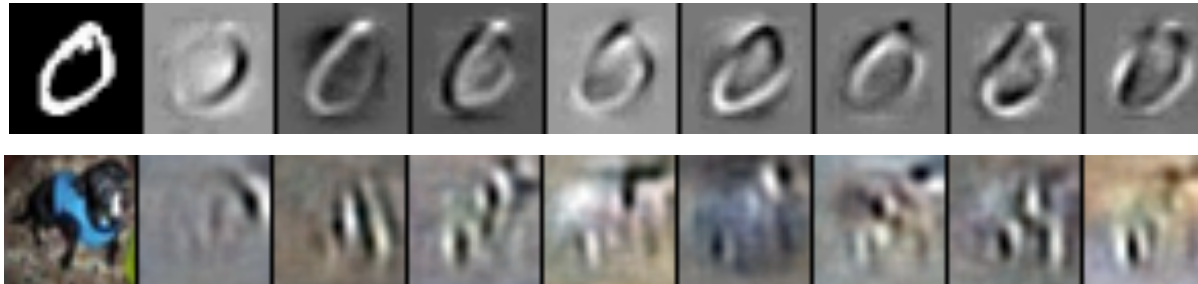
Algorithm:

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%	0.81%

- Semi-sup.

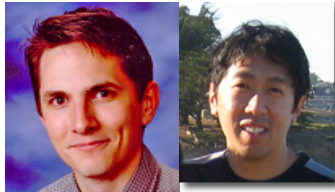
	NN	SVM	CNN	TSVM	DBN-rNCA	EmbedNN	CAE	MTC
100	25.81	23.44	22.98	16.81	-	16.86	13.47	12.03
600	11.44	8.85	7.68	6.16	8.7	5.97	6.3	5.13
1000	10.7	7.77	6.45	5.38	-	5.73	4.77	3.64
3000	6.04	4.21	3.35	3.45	3.3	3.59	3.22	2.57

- Forest (500k examples)

SVM	Distributed SVM	MTC
4.11%	3.46%	3.13%

Inference and Explaining Away

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



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

Sparse Coding

(Olshausen et al 97)



- Directed graphical model:

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

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

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

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

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

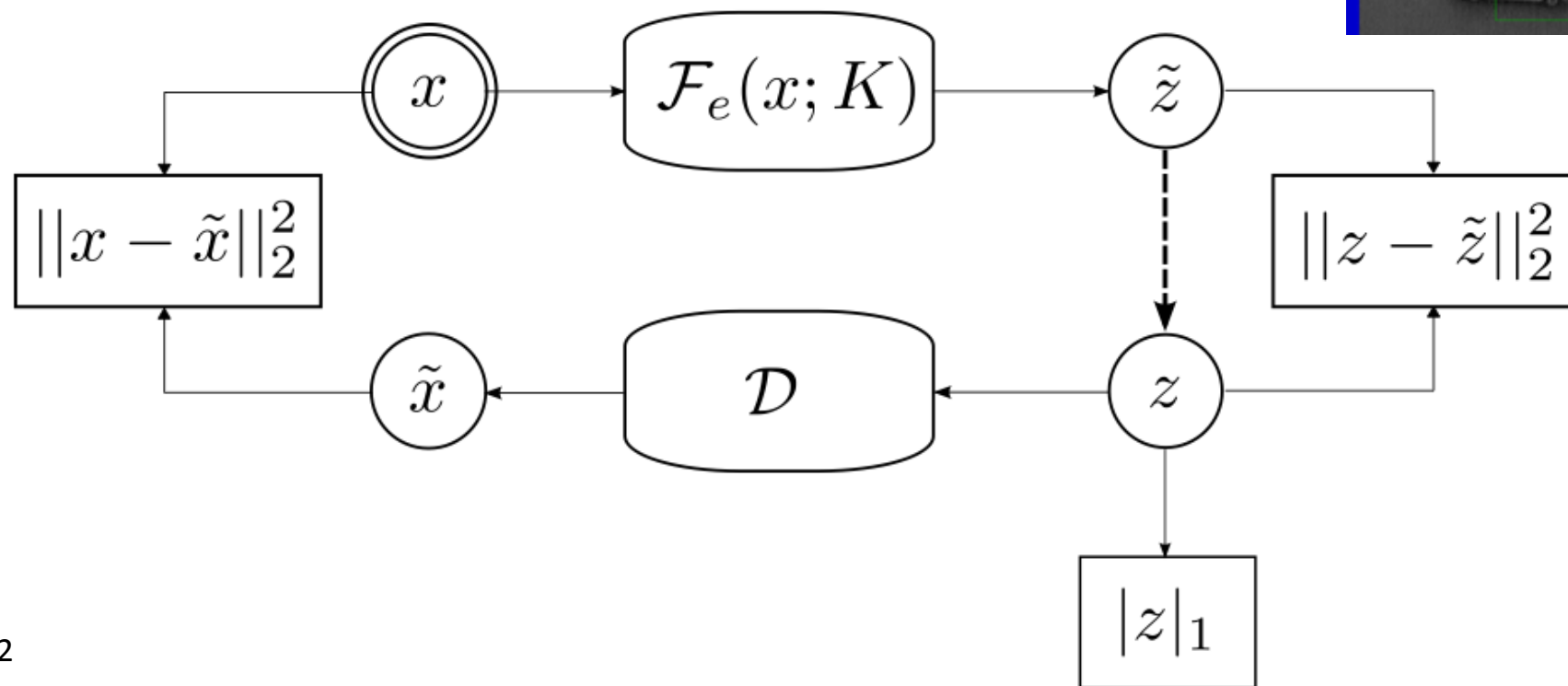
Predictive Sparse Decomposition



- Approximate the inference of sparse coding by an encoder:

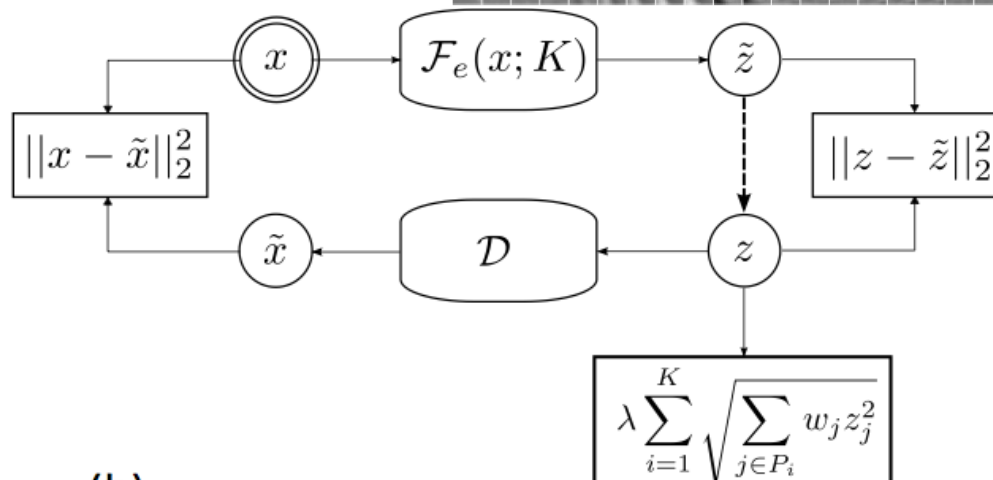
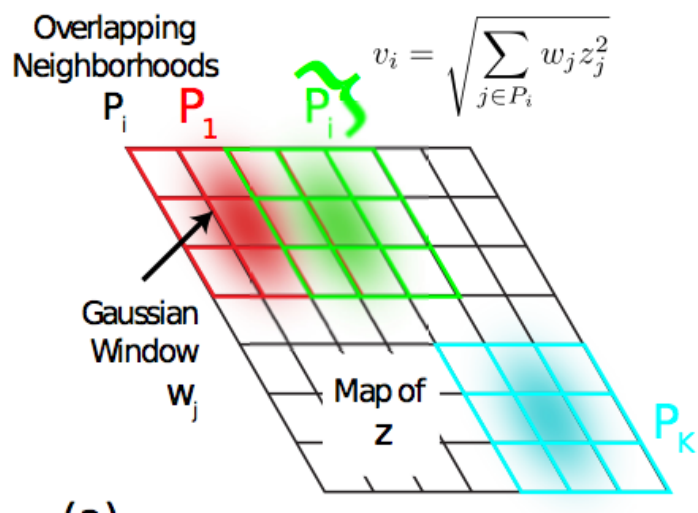
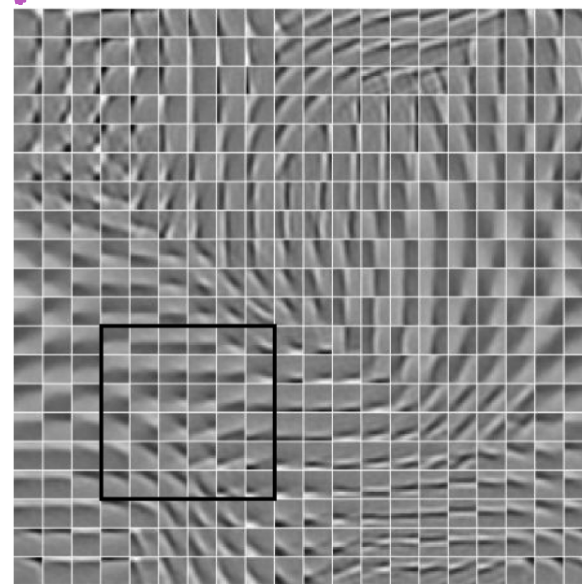
Predictive Sparse Decomposition (Kavukcuoglu et al 2008)

- Very successful applications in machine vision with convolutional architectures



Predictive Sparse Decomposition

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



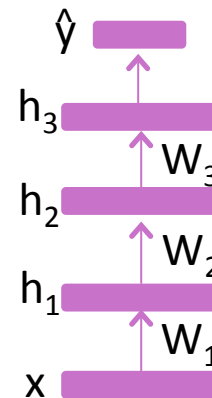
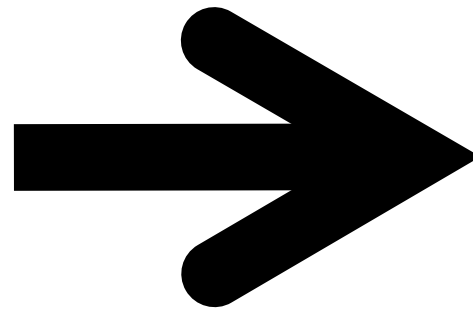
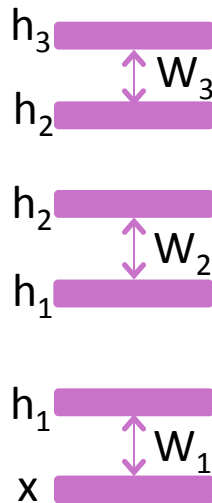
Deep Variants

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 helps a lot
- Helps most the layers further away from the target
- Not just an effect of unsupervised prior
- Jointly training all the levels of a deep architecture is difficult
- Initializing using a **level-local learning algorithm** is a useful trick

Stack of RBMs / AEs → Deep MLP

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

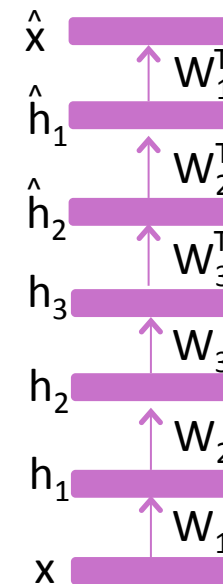
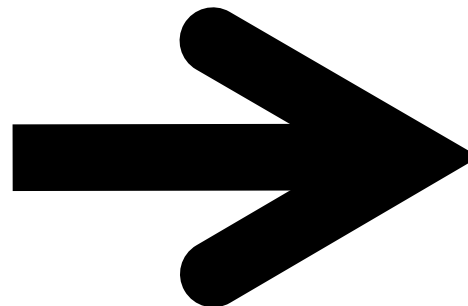
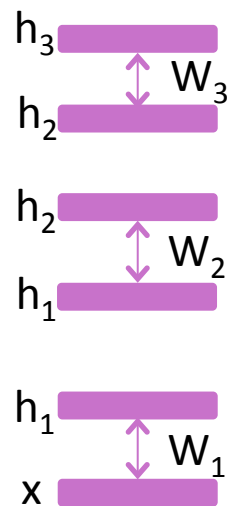


Stack of RBMs / AEs → Deep Auto-Encoder



(Hinton & Salakhutdinov 2006)

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



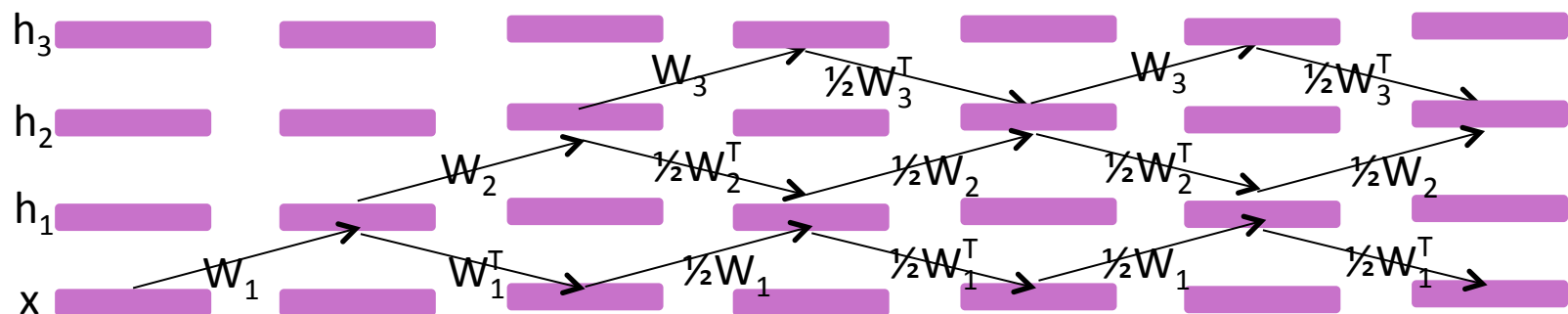
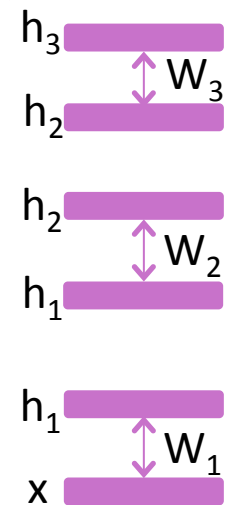
Stack of RBMs / AEs

→ Deep Recurrent Auto-Encoder

(Savard 2011)



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

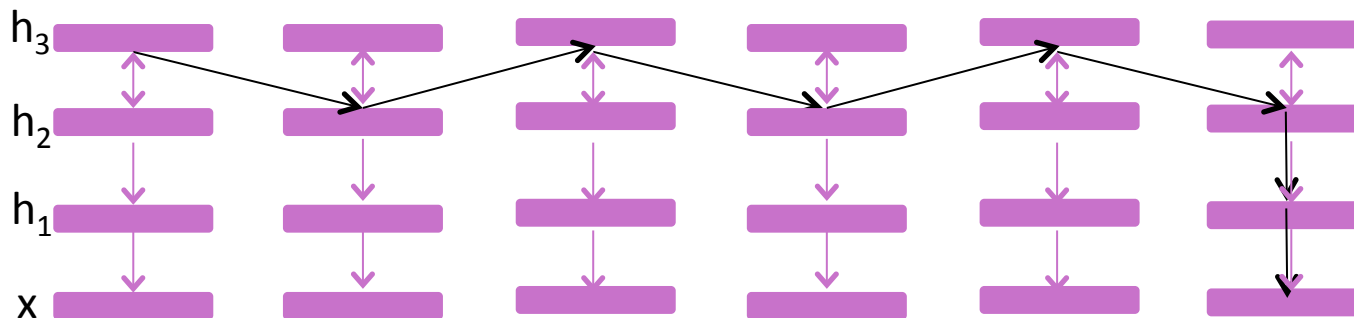


Stack of RBMs → Deep Belief Net



(Hinton et al 2006)

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



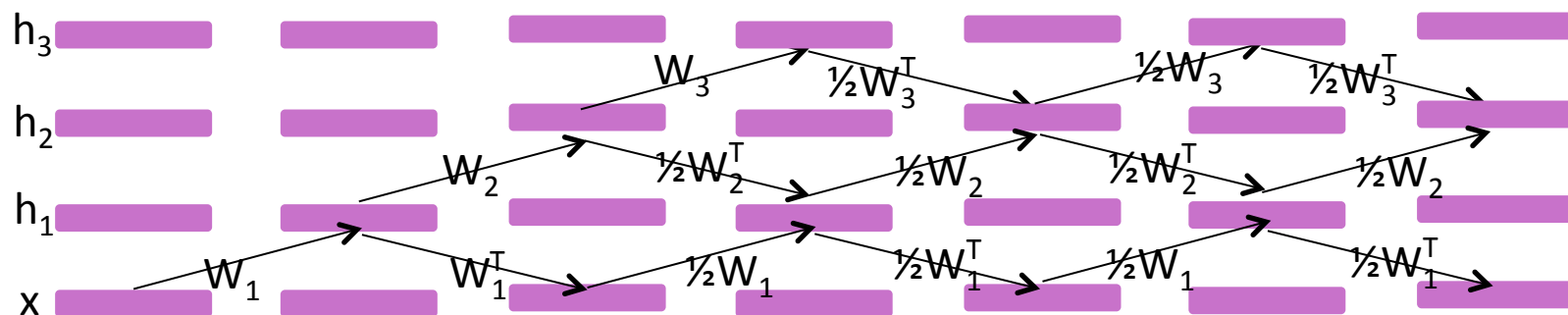


Stack of RBMs

→ Deep Boltzmann Machine

(Salakhutdinov & Hinton AISTATS 2009)

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

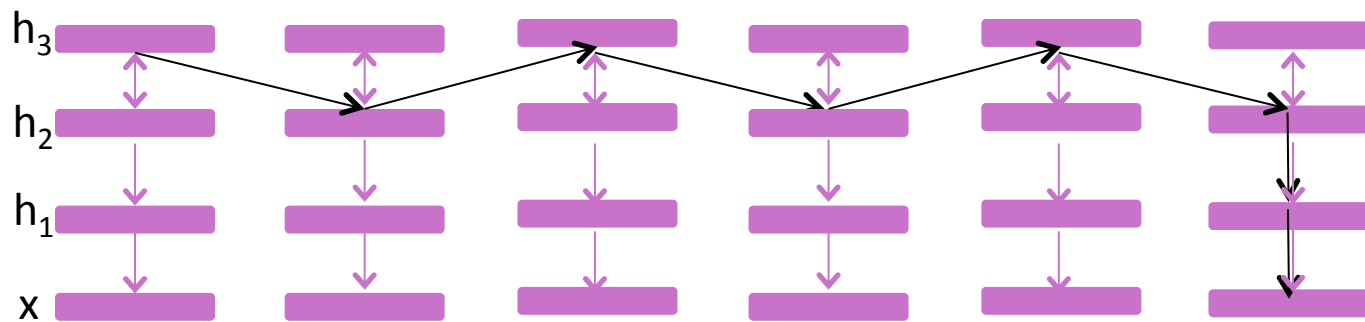


Stack of Auto-Encoders → Deep Generative Auto-Encoder

(Rifai et al ICML 2012)



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

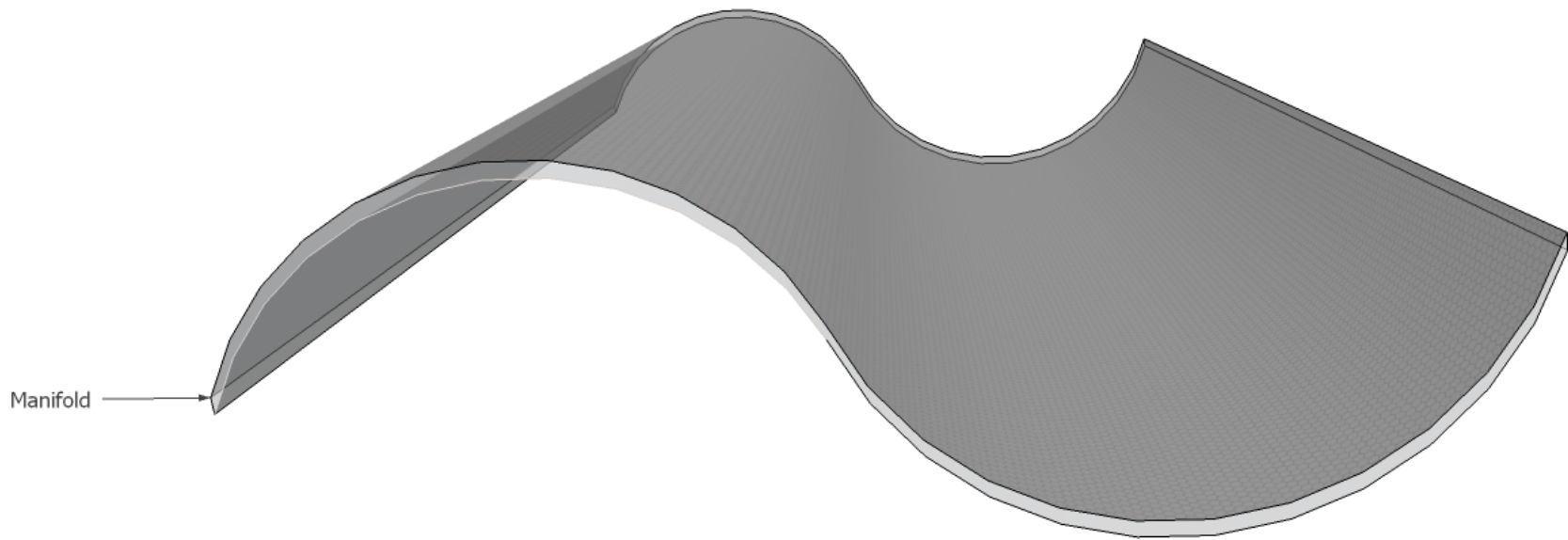


Manifold Learning Interpretation Allows Sampling from Auto-Encoders

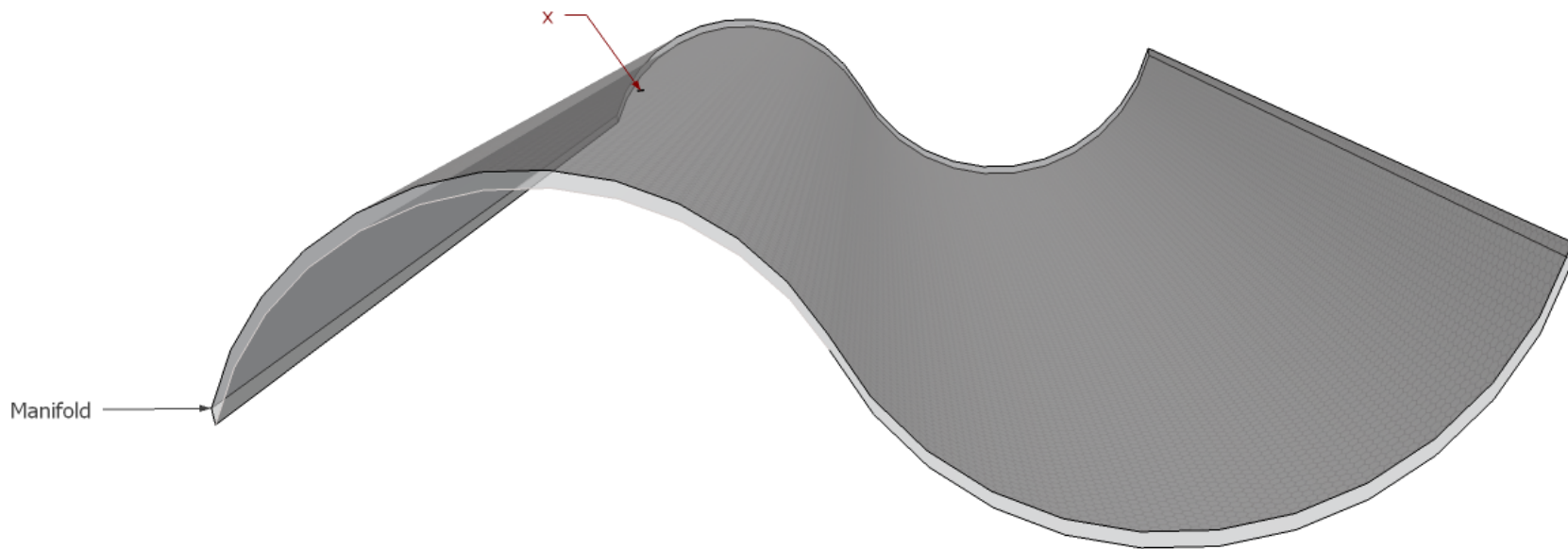
- Reconstruction function captures geometry of the input distribution
- $reconstruction(x)-x$ points towards high-density (score)
- Jacobian of $reconstruction(x)$ has large singular values in directions of local factors of variation (manifold tangents)
- Gives rise to an implicit density estimator and a **sampling algorithm** for contractive and denoising auto-encoders (Rifai et al ICML 2012)



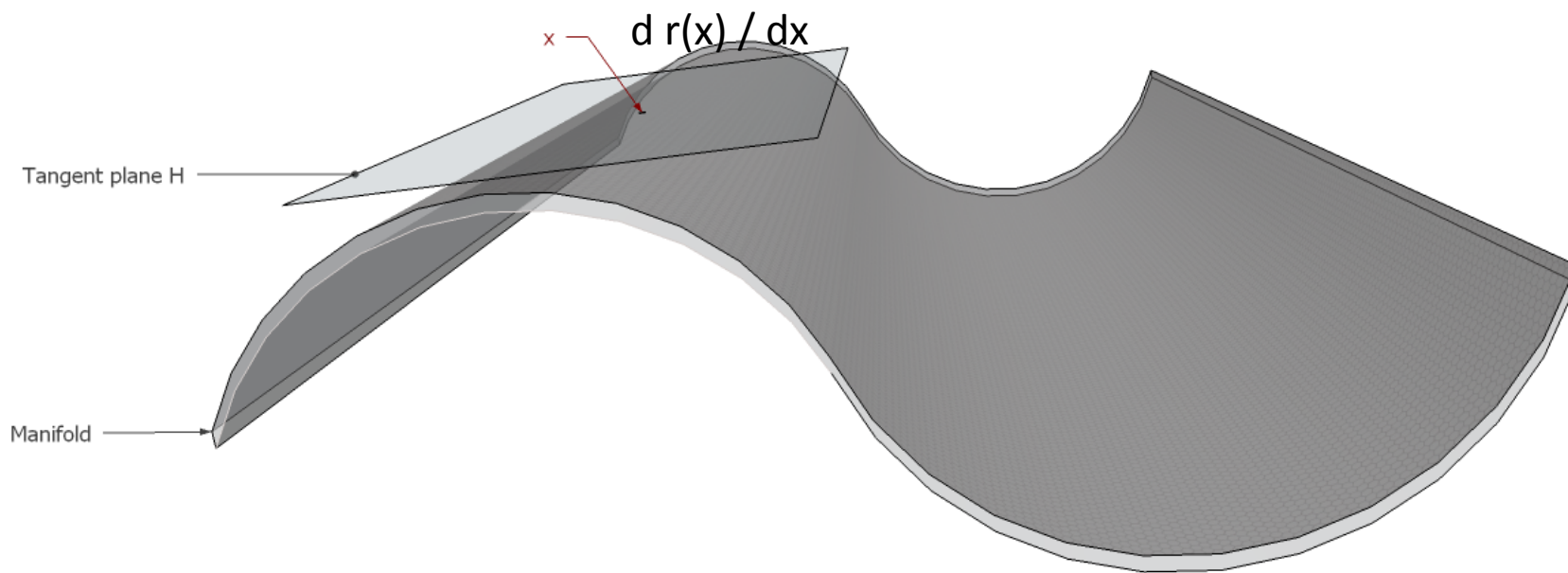
Sampling from a Regularized Auto-Encoder



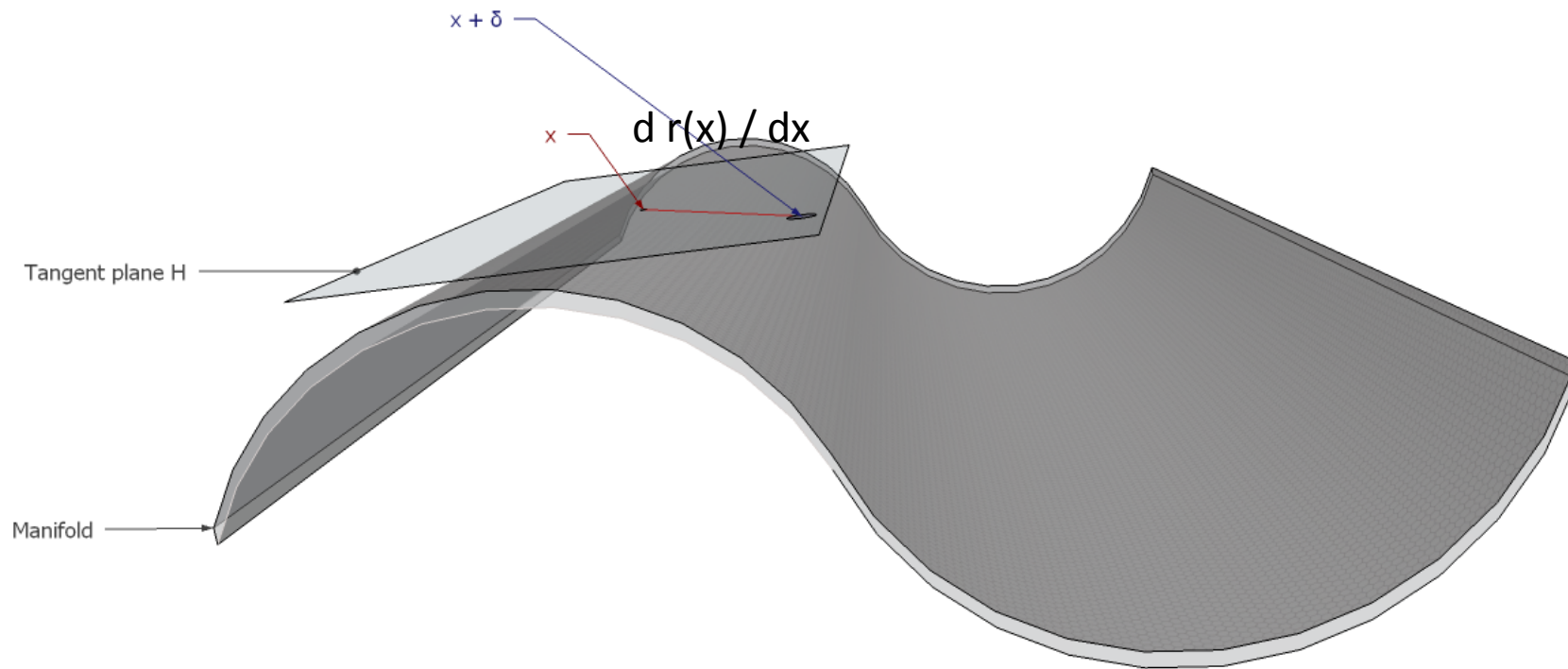
Sampling from a Regularized Auto-Encoder



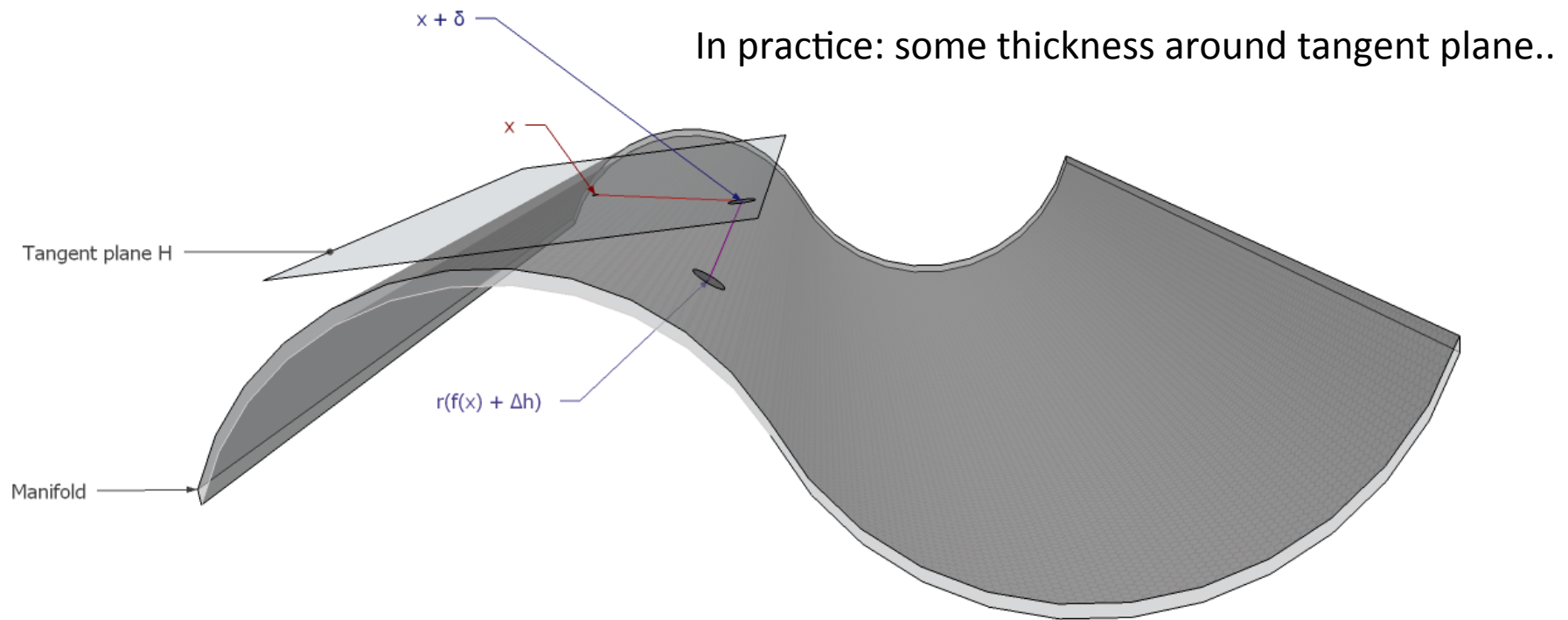
Sampling from a Regularized Auto-Encoder



Sampling from a Regularized Auto-Encoder



Sampling from a Regularized Auto-Encoder



Samples from a 2-Level DAE

- TFD



- MNIST



Samples from a 2-level CAE (ICML 2012)

Table 1. Log-Likelihoods from Parzen density estimator using 10000 samples from each model

	DBN-2	CAE-2
TFD	1908.80 ± 65.94	2110.09 ± 49.15
MNIST	137.89 ± 2.11	121.17 ± 1.59



- Not using local covariance estimator, just isotropic noise: **bad**



MCMC Asymptotic Distribution: Uncountable Gaussian Mixture

- Each step samples next x from Gaussian with mean and covariance a function of previous \tilde{x}
- Asymptotic distribution (if exists):

$$\pi(x) = \int \pi(\tilde{x}) \mathcal{N}(x; \mu(\tilde{x}), \Sigma(\tilde{x})) d\tilde{x}$$

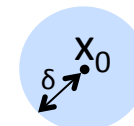
= uncountable gaussian mixture with weights = the density itself

- Thm: If $\Sigma(x)$ is full-rank and $\mu(x)$ in bounded region, then π exists.

Consistency: Samples \leftrightarrow Local Moments

(Bengio et al 2012, arXiv paper, “Implicit Density Estimation by Local Moment Matching to Sample from Auto-Encoders”)

- Inside-ball density: $p_\delta(x|x_0) = \frac{p(x)1_{\|x-x_0\|<\delta}}{Z(x_0)}$
- Ball size $\delta \rightarrow 0$ around each x_0 , MCMC steps of size $\sigma \ll \delta$



$$\begin{aligned} m_0 = E_\pi[x|x_0] &= \frac{1}{Z(x_0)} \int_x x \int_{\tilde{x}} p(\tilde{x}) \mathcal{N}(x; \mu(\tilde{x}), \Sigma(\tilde{x})) d\tilde{x} 1_{\|x-x_0\|<\delta} dx \\ &= \frac{1}{Z(x_0)} \int_{\tilde{x}} p(\tilde{x}) \int_{\|x-x_0\|<\delta} x \mathcal{N}(x; \mu(\tilde{x}), \Sigma(\tilde{x})) dx d\tilde{x}. \\ &\approx \int_{\tilde{x}} \frac{p(\tilde{x})}{Z(x_0)} 1_{\|\mu(\tilde{x})-x_0\|<\delta} \mu(\tilde{x}) d\tilde{x} \\ &\approx \mathbb{E}[\mu(x)|x_0] \end{aligned}$$

- i.e. the **local mean** $m_0 \approx$ expected value of MCMC mean in the ball, and similarly for **local covariance** C_0 & MCMC covariance.
- *Step size σ controls quality of approximation, which corresponds to a smooth of the estimated density.*

Consistency: Non-Parametric / Asymptotic Minimizer of Criterion

- Training criterion rewritten:

$$\begin{aligned}\mathcal{L}_{\text{global}} &= \int p(x_0) \left(\|x_0 - r(x_0)\|^2 + \alpha \left\| \frac{\partial r(x_0)}{\partial x_0} \right\|_F^2 \right) dx_0 \\ &= \lim_{\delta \rightarrow 0} \int p(x_0) \left(\left(\int_x \|x - r(x)\|^2 p_\delta(x|x_0) dx \right) + \alpha \left\| \frac{\partial r(x_0)}{\partial x_0} \right\|_F^2 \right) dx_0\end{aligned}$$

- Local (non-parametric) parametrization around x_0

$$r(x) = r(x_0) + \left. \frac{\partial r}{\partial x} \right|_{x_0} (x - x_0) = r_0 + J_0(x - x_0)$$

$$\mathcal{L}_{\text{local}}(x_0, \delta) = \int_x \|x - (r_0 + J_0(x - x_0))\|^2 p_\delta(x|x_0) dx + \alpha \|J_0\|_F^2$$

$$\mathcal{L}_{\text{global}} = \lim_{\delta \rightarrow 0} \int p(x_0) \mathcal{L}_{\text{local}}(x_0, \delta) dx_0$$

Consistency: Non-Parametric / Asymptotic Minimizer of Criterion

- Solving:
$$\frac{\partial \mathcal{L}_{\text{local}}(x_0, \delta)}{\partial r_0} = 0$$
$$\frac{\partial \mathcal{L}_{\text{local}}(x_0, \delta)}{\partial J_0} = 0$$

yields:

$$r_0 = (I - J_0)m_0 + J_0x_0$$

$$J_0 = C_0(\alpha I + C_0)^{-1}.$$

i.e. when $\delta \rightarrow 0$ (i.e. $J_0 \rightarrow 0$), \asymp means lhs / rhs $\rightarrow 1$:

$$r_0 \asymp m_0$$

$$J_0 \asymp \alpha^{-1} C_0$$

- **Reconstruction and its Jacobian estimate local mean & covariance**

Implicit Density Estimation

- In general, no explicit analytic formulation of the estimated density, only of its local moments and 1st & 2nd derivatives
- Can obtain samples by MCMC (of a smooth of estimated density)
- Alternatively, can parametrize $r(x)-x = \text{derivative of an energy function } energy(x)$ which provides an explicit analytic formulation of the estimated density.
- **We have avoided the partition function and introduced a novel(?) alternative to maximum likelihood**


AE sampling: open questions

- Effects of parametric non-asymptotic setting?
- Training energy-based models as regularized AE
- Why better results when training as CAE vs DAE?

Part 3

Practice, Issues, Questions

Deep Learning Tricks of the Trade

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

Stochastic Gradient Descent (SGD)

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

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

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

Learning Rates

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

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

with hyper-parameters ϵ_0 and τ .

Early Stopping

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

Long-Term Dependencies

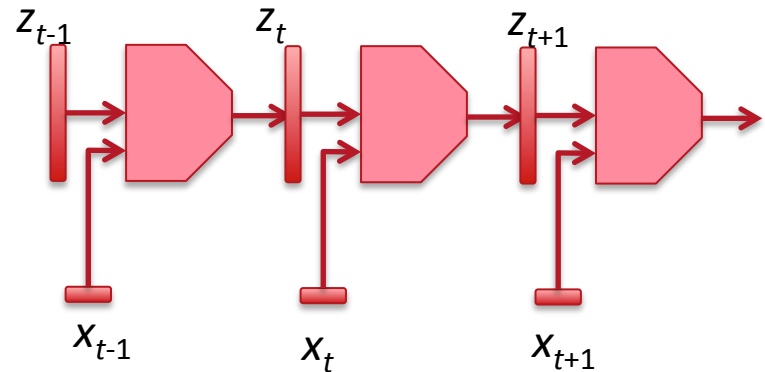


- In very deep networks such as **recurrent networks** (or possibly recursive ones), the gradient is a product of Jacobian matrices, each associated with a step in the forward computation. This can become very small or very large quickly [Bengio et al 1994], and the locality assumption of gradient descent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

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

Normalized Initialization to Achieve Unity-Like Jacobian

Assuming $f'(act=0)=1$

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:**

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

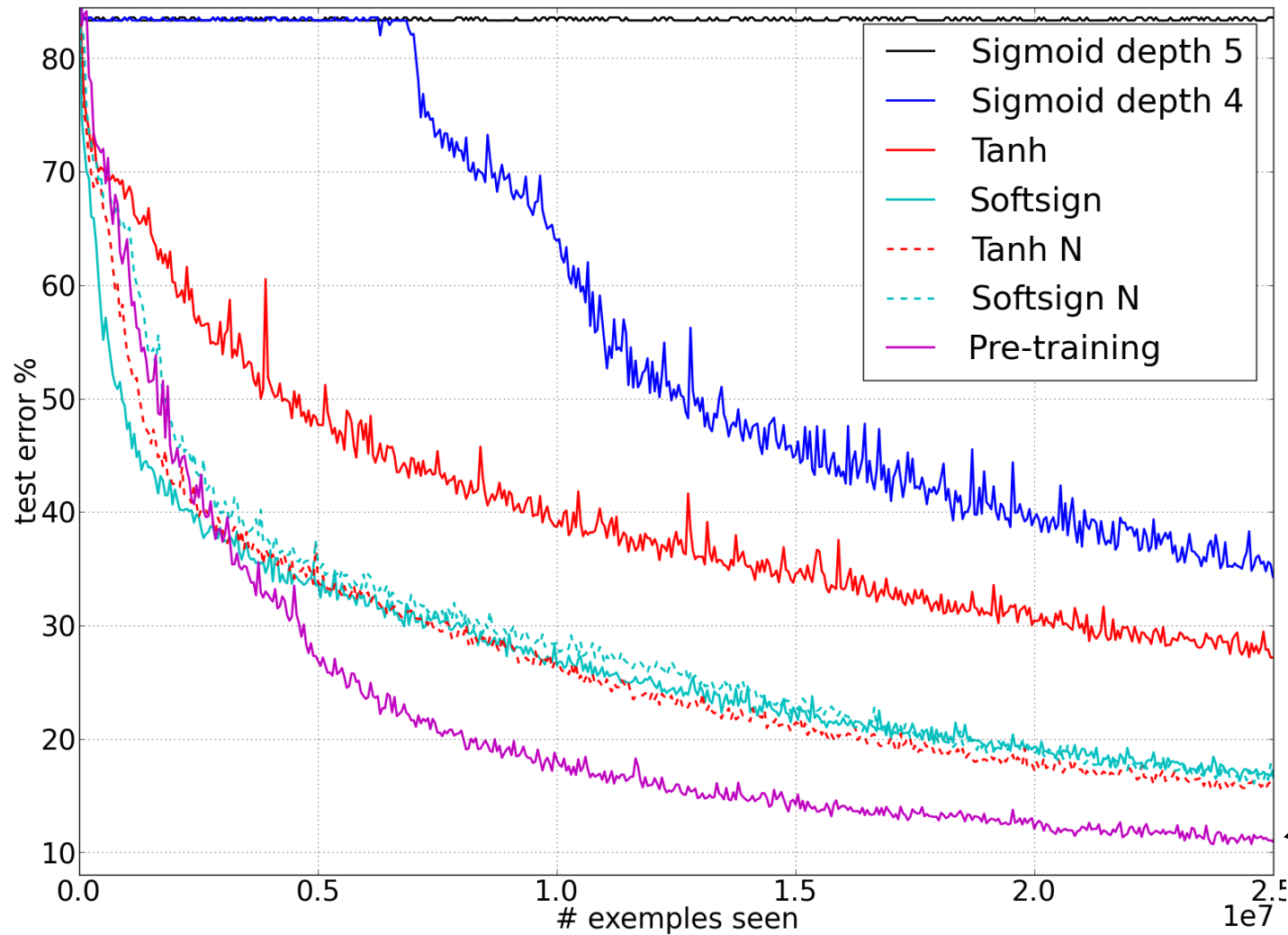
Possible compromise:

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

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



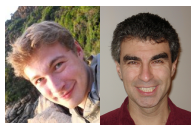
Parameter Initialization

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

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

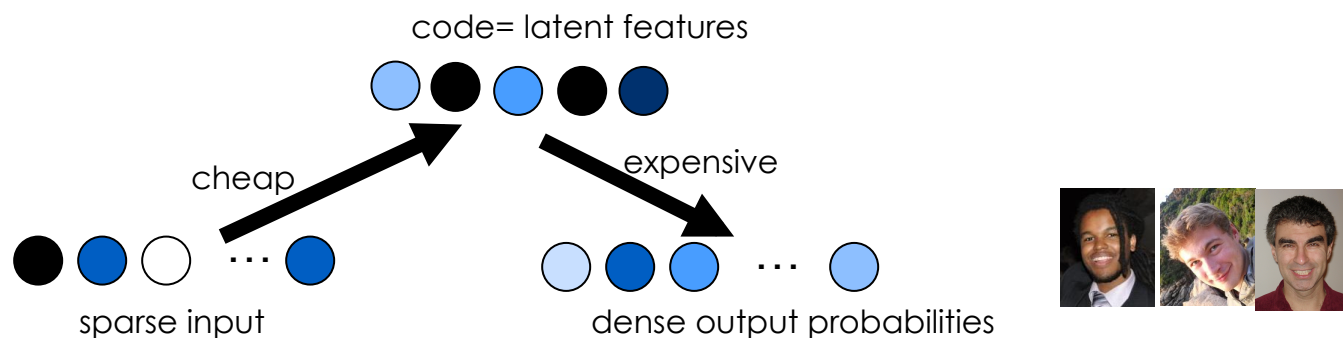
for tanh units (and 4x bigger for sigmoid units)

(Glorot & Bengio AISTATS 2010)



Handling Large Output Spaces

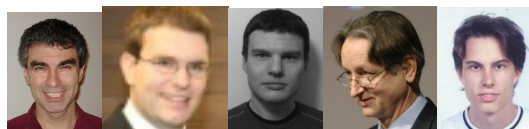
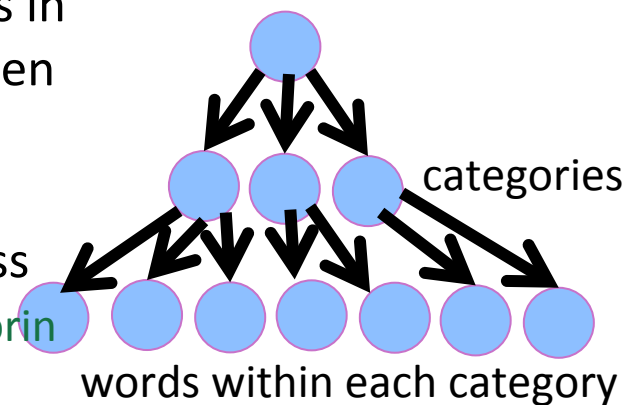
- Auto-encoders and RBMs reconstruct the input, which is sparse and high-dimensional; Language models have huge output space.



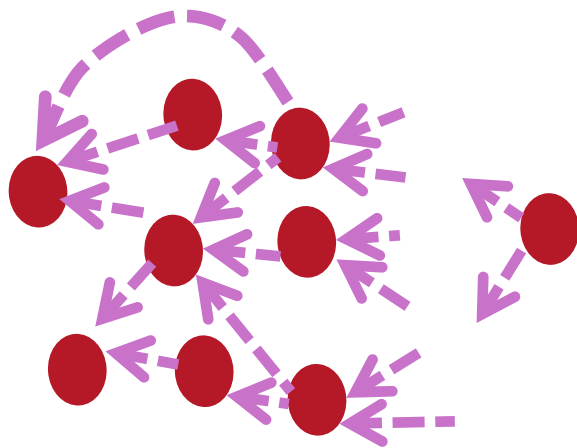
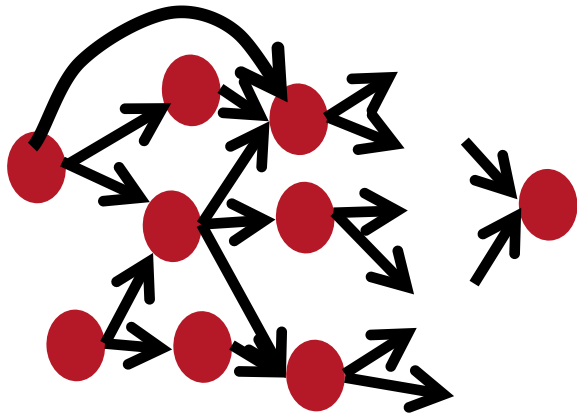
- (Dauphin et al, ICML 2011) Reconstruct the non-zeros in the input, and reconstruct as many randomly chosen zeros, + importance weights



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



Automatic Differentiation



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

(Bergstra et al SciPy'2010)



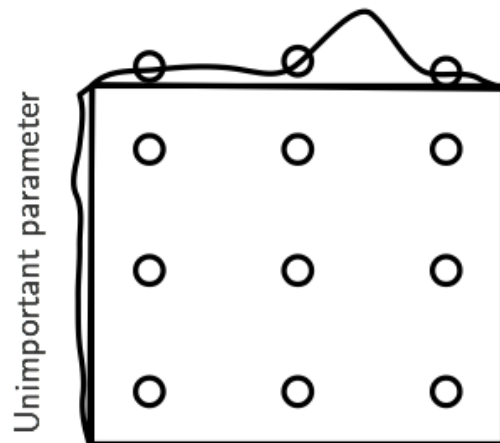
Random Sampling of Hyperparameters

(Bergstra & Bengio 2012)



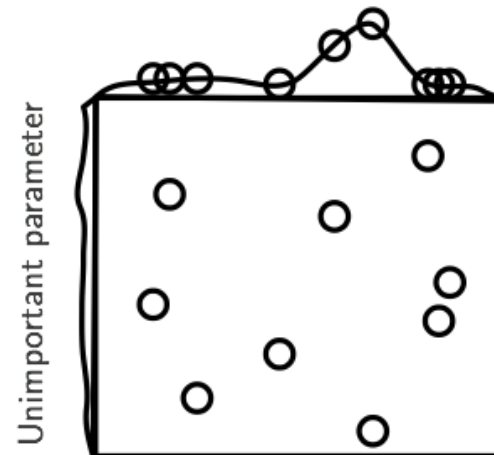
- Common approach: manual + grid search
- Grid search over hyperparameters: simple & wasteful
- Random search: simple & efficient
 - Independently sample each HP, e.g. $\text{l.rate} \sim \exp(\text{U}[\log(.1), \log(.0001)])$
 - Each training trial is iid
 - If a HP is irrelevant grid search is wasteful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

Grid Layout



Important parameter

Random Layout



Important parameter

Issues and Questions

Why is Unsupervised Pre-Training Working So Well?

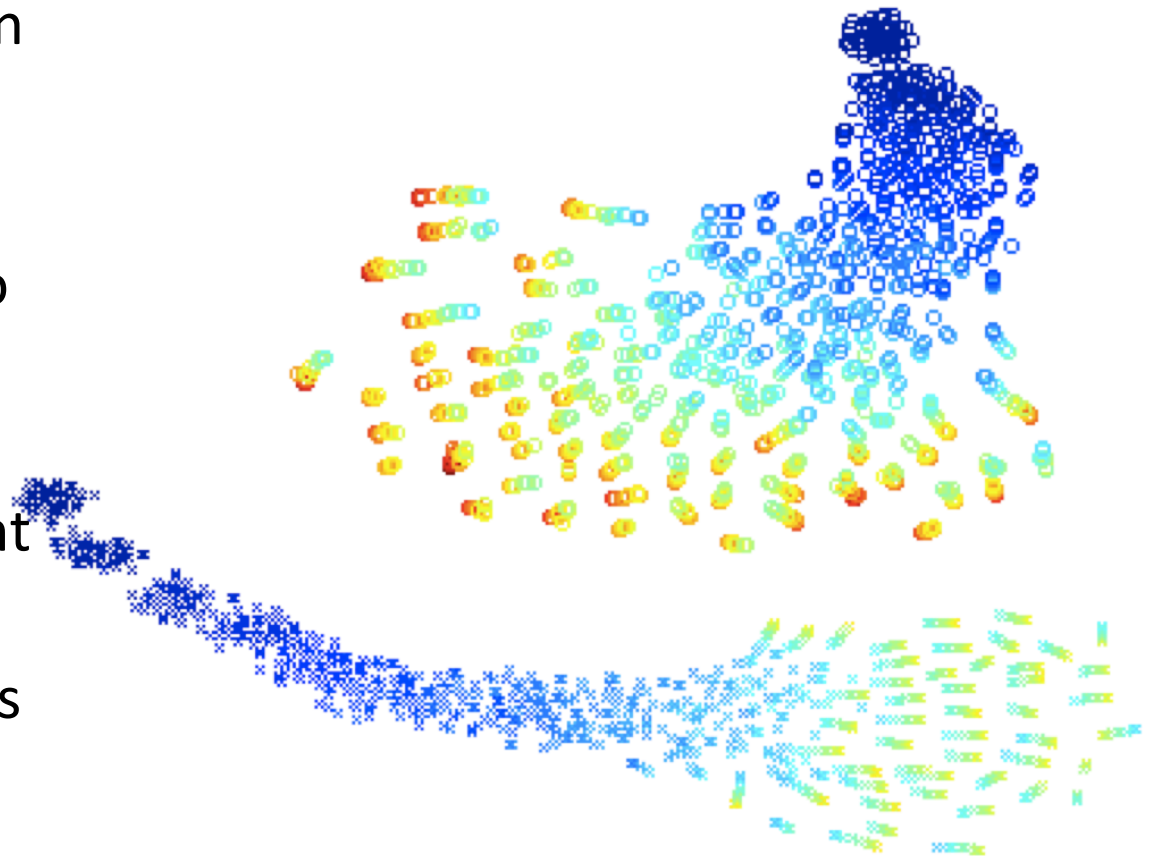
- Regularization hypothesis:
 - Unsupervised component forces model close to $P(x)$
 - Representations good for $P(x)$ are good for $P(y|x)$
- Optimization hypothesis:
 - Unsupervised initialization near better local minimum of $P(y|x)$
 - Can reach lower local minimum otherwise not achievable by random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



(Erhan et al JMLR 2010)

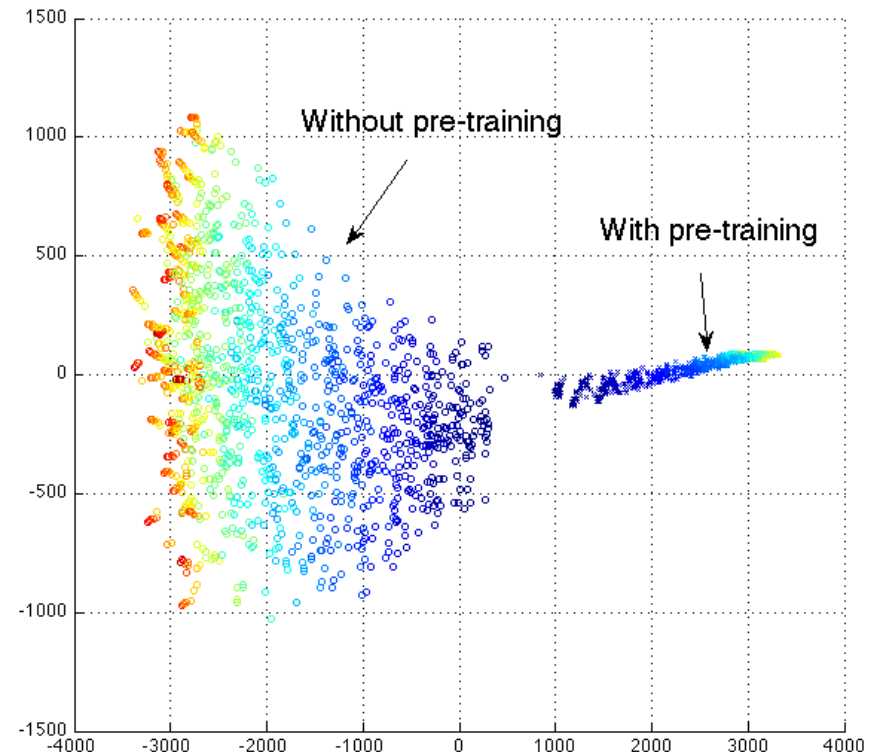
Learning Trajectories in Function Space

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



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)



Inference Challenges

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



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

Dealing with Inference

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

Dealing with a Partition Function

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

Score Matching

(Hyvarinen 2005)

- Score of model p : $d \log p(\mathbf{x})/d\mathbf{x}$ **does not contain partition fn Z**

- Matching score of p to target score: **?**

$$\mathbb{E}_{q(\mathbf{x})} \left[\frac{1}{2} \left\| \frac{\partial \log p(\mathbf{x})}{\partial \mathbf{x}} - \frac{\partial \log q(\mathbf{x})}{\partial \mathbf{x}} \right\|^2 \right]$$

- Hyvarinen shows it equals

$$\mathbb{E}_{q(\mathbf{x})} \left[\frac{1}{2} \left\| \frac{\partial \log p(\mathbf{x})}{\partial \mathbf{x}} \right\|^2 + \sum_i \frac{\partial^2 \log p(\mathbf{x})}{\partial \mathbf{x}_i^2} \right] + \text{const}$$

- and proposes to minimize corresponding empirical mean
- Shown to be asymptotically unbiased to estimate parameters
- Note: for GRBM, 1st term is squared reconstruction error and 2nd term looks like contractive penalty $\sum_{ij} h_i(1-h_i) W_{ij}^2$



Denoising Auto-Encoders doing Score Matching on Gaussian RBMs

- clean input - corrupted input = direction of increasing log-likelihood

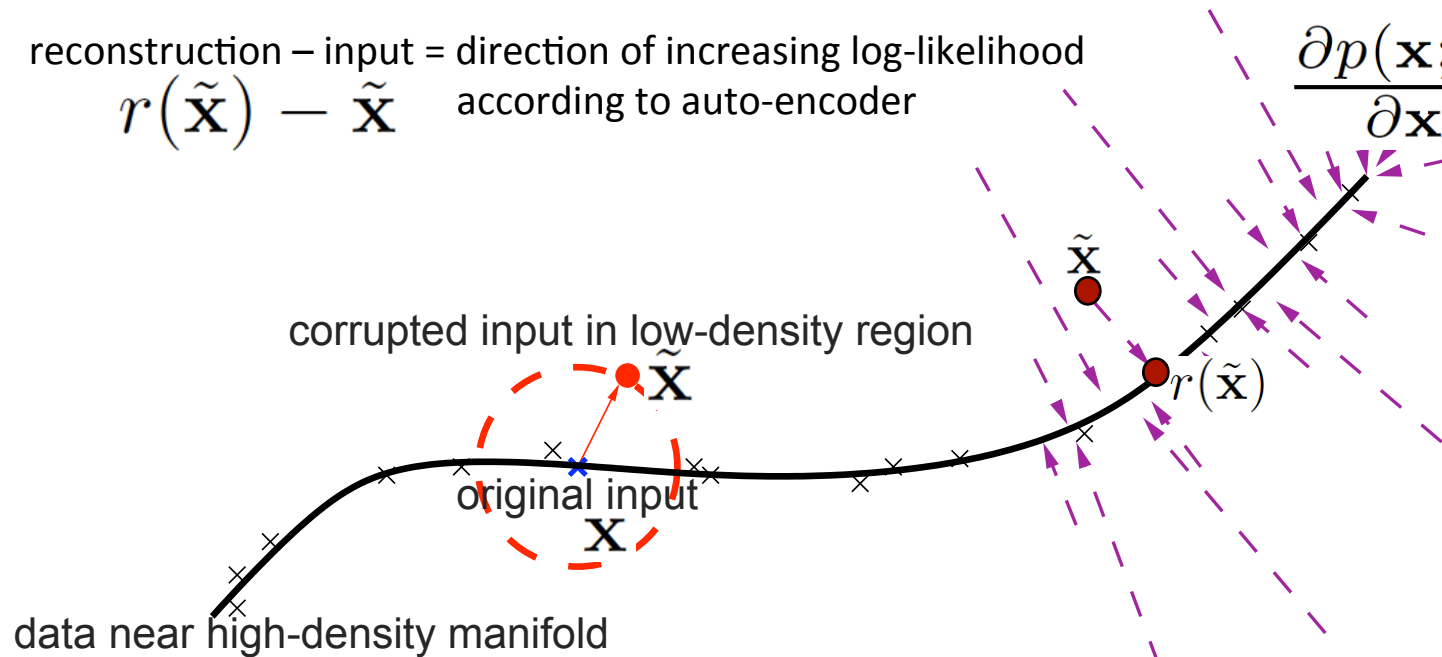
$$\mathbf{x} - \tilde{\mathbf{x}}$$

$$\frac{\partial \log q_{\sigma}(\tilde{\mathbf{x}}|\mathbf{x})}{\partial \tilde{\mathbf{x}}}$$

- reconstruction - input = direction of increasing log-likelihood according to auto-encoder

$$r(\tilde{\mathbf{x}}) - \tilde{\mathbf{x}}$$

$$\frac{\partial p(\mathbf{x};\theta)}{\partial \mathbf{x}}$$



- Denoising error = $\| (r(\tilde{\mathbf{x}}) - \tilde{\mathbf{x}}) - (\mathbf{x} - \tilde{\mathbf{x}}) \|^2 = \| r(\tilde{\mathbf{x}}) - \mathbf{x} \|^2$

Denoising Auto-Encoders doing Score Matching on Gaussian RBMs

- Gaussian DAE reconstruction:

$$r(\tilde{\mathbf{x}}) = \mathbf{W}^T h(\tilde{\mathbf{x}}) + \mathbf{c} = \mathbf{W}^T \text{sigm}(\mathbf{W}\tilde{\mathbf{x}} + \mathbf{b}) + \mathbf{c}$$

- Corresponds to gradient of free energy of Gaussian RBM
- Any (free) energy function with x^2 term gives rise to score function that can be written as proportional to $r(x)-x$ (=residual). **Recent research shows this is true for any energy fn**
- Not all DAEs have reconstruction residual = a derivative (most previous DAEs with binomial KL divergence reconstruction error)
- See also ([Swersky 2010](#)), thesis on link between ordinary auto-encoder reconstruction error & Score Matching

Contrastive Sampling of Negative Examples

- (Collobert & Weston ICML 2008)
- (Bordes et al AAI 2011, AISTATS 2012)
- Similar to wsabie (Weston et al, MLJ 2010)

$$\sum_{x \in \mathcal{D}} \sum_{\tilde{x} \sim Q(\tilde{x}|x)} \max(\mathcal{E}(x) - \mathcal{E}(\tilde{x}) + 1, 0)$$

Positive example Negative example

pull down push up

energy function

- In those cases, the negative example \tilde{x} is obtained by uniformly sampling one of the elements of \bar{x} and keeping the rest fixed.

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?

Poor Mixing: Depth to the Rescue

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

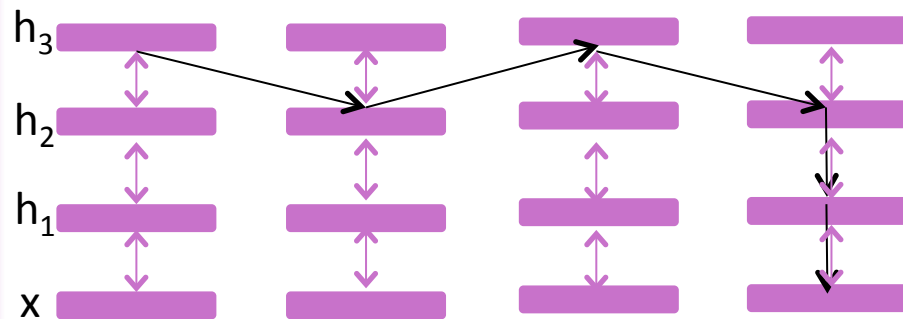
Layer abstract level



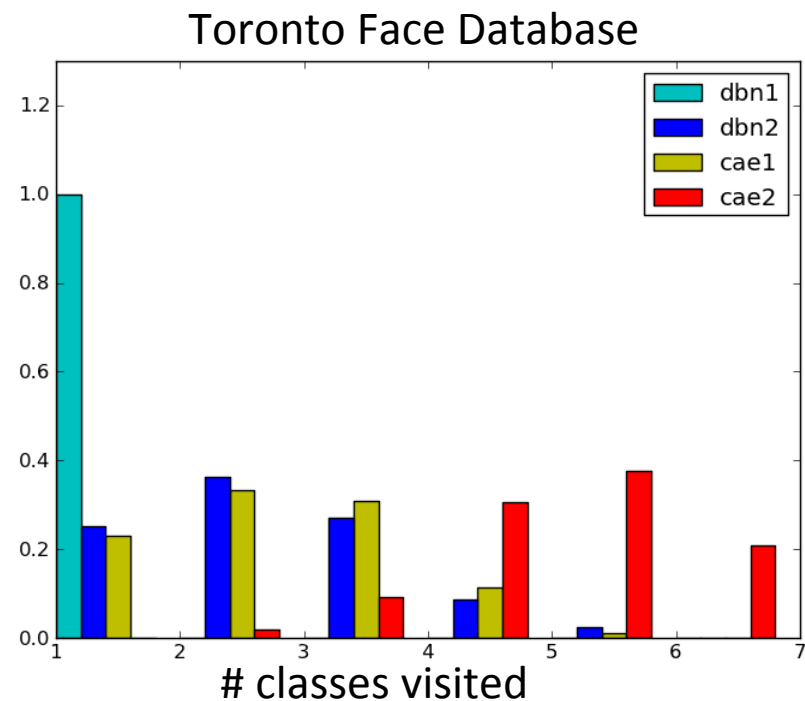
Points on the interpolating line between two classes, at different levels of representation

Poor Mixing: Depth to the Rescue

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



192



Regularized AE: MCMC Miracles?

- Virtually no burn-in waste with Denoising AE, trained to map random configurations to plausible ones in 1 step (very few necessary in practice)
- Tempering-like effect by controlling step size σ (in manifold directions) trades off mixing speed with accuracy (more math needed here!)
- Fast mode mixing if sampling at higher levels

Other reasons why regularized auto-encoders are interesting alternatives

- Easy “inference” (can have iterative inference with lateral connections, but not considered as an approximation to the right thing)
 - No partition function (and associated approximations)
 - No negative feedback loop between sampling and learning
- **do we actually need an explicit probabilistic model?**

More Open Questions

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

Natural Language Processing (NLP) Applications

(parts coming from ACL'2012 tutorial on Deep Learning for NLP, with Richard Socher and Chris Manning)

Deep Learning models have already achieved impressive results for NLP

Neural Language Model

[Mikolov et al. Interspeech 2011]



Model \ WSJ task	Eval WER
KN5 Baseline	17.2
Discriminative LM	16.9
Recurrent NN combination	14.4

MSR MAVIS Speech System

[Dahl et al. 2012; Seide et al. 2011; following Mohamed et al. 2011]



“The algorithms represent the first time a company has released a deep-neural-networks (DNN)-based speech-recognition algorithm in a commercial product.”

Acoustic model & training	Recog \ WER	RT03S FSH	Hub5 SWB
GMM 40-mix, BMMI, SWB 309h	1-pass -adapt	27.4	23.6
CD-DNN 7 layer x 2048, SWB 309h	1-pass -adapt	18.5 (-33%)	16.1 (-32%)
GMM 72-mix, BMMI, FSH 2000h	k-pass +adapt	18.6	17.1

Existing NLP Applications

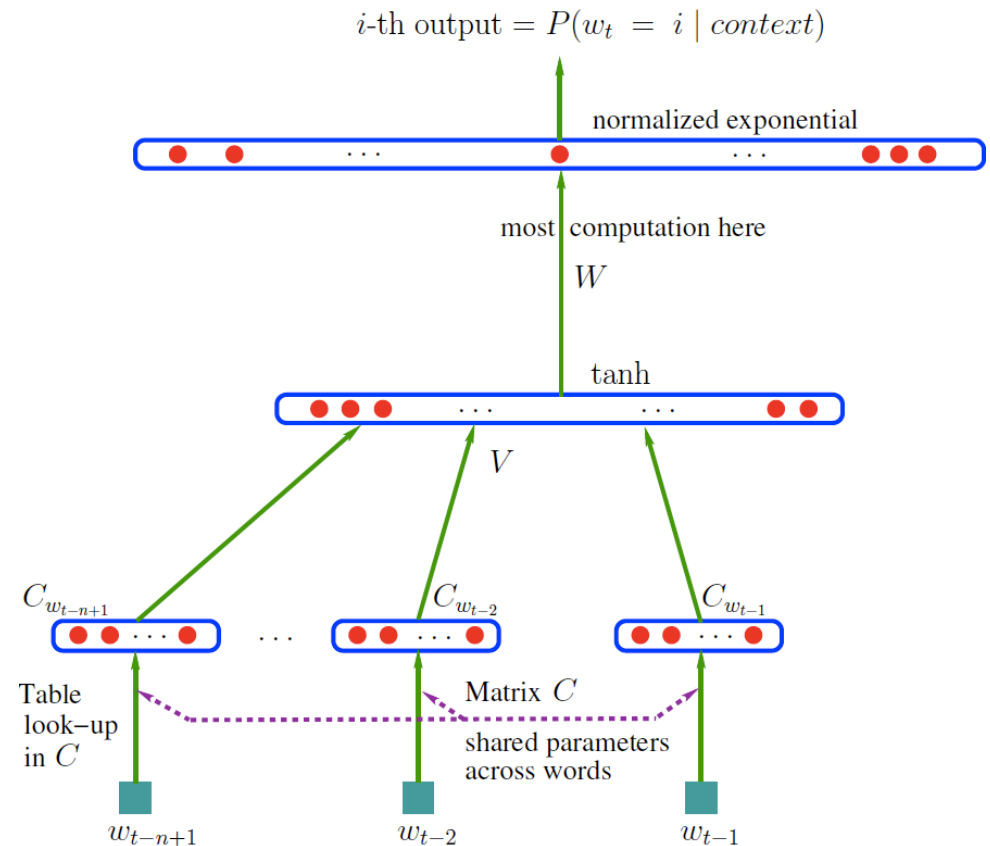
- Language Modeling (Speech Recognition, Machine Translation)
- Acoustic Modeling
- Part-Of-Speech Tagging
- Chunking
- Named Entity Recognition
- Semantic Role Labeling
- Parsing
- Sentiment Analysis
- Paraphrasing
- Question-Answering
- Word-Sense Disambiguation

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
- Generalizes to sequences of words that are semantically similar to training sequences



Language Modeling

- Predict $P(\text{next word} \mid \text{previous word})$
- Gives a probability for a longer sequence
- Applications to Speech, Translation and Compression
- Computational bottleneck: large vocabulary V means that computing the output costs $\# \text{hidden units} \times |V|$.

The standard word representation

The vast majority of rule-based **and** statistical NLP work regards words as atomic symbols: *hotel, conference, walk*

In vector space terms, this is a vector with one 1 and a lot of zeroes

$[0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0]$

Dimensionality: 20K (speech) – 50K (PTB) – 500K (big vocab) – 13M (Google 1T)

We call this a “*one-hot*” representation. Its problem:

motel $[0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0]$ AND
hotel $[0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0]$ = 0

Distributional similarity based representations

You can get a lot of value by representing a word by means of its neighbors

“You shall know a word by the company it keeps”

(J. R. Firth 1957: 11)

One of the most successful ideas of modern statistical NLP

government debt problems turning into banking crises as has happened in

saying that Europe needs unified banking regulation to replace the hodgepodge

↖ These words will represent *banking* ↗

You can vary whether you use local or large context to get a more syntactic or semantic clustering

Class-based (hard) and soft clustering word representations

Class based models learn word classes of similar words based on distributional information (\sim class HMM)

- Brown clustering (Brown et al. 1992)
- Exchange clustering (Martin et al. 1998, Clark 2003)
- Desparsification and great example of unsupervised pre-training

Soft clustering models learn for each cluster/topic a distribution over words of how likely that word is in each cluster

- Latent Semantic Analysis (LSA/LSI), Random projections
- Latent Dirichlet Analysis (LDA), HMM clustering

Neural word embeddings as a distributed representation

Similar idea, but think of each dimension as an attribute, not as a cluster membership

Combine vector space semantics with the prediction of probabilistic models (Bengio et al. 2003, Collobert & Weston 2008, Turian et al. 2010)

In all of these approaches, including deep learning models, a word is represented as a dense vector (TODO: sparsity)

linguistics =

$$\begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$$

Neural word embeddings - visualization



Advantages of the neural word embedding approach

Compared to a method like LSA, neural word embeddings can become **more meaningful** through adding supervision from one or multiple tasks

For instance, sentiment is usually not captured in unsupervised word embeddings but can be in neural word vectors

We can build representations for large linguistic units

See below

Contrastive Sampling of Negative Examples (Collobert et al. JMLR 2011)

Idea: A word and its context is a positive training sample; a random word in that same context gives a negative training sample:

+ cat chills on a mat - cat chills Jeju a mat

Similar: Implicit negative evidence in Contrastive Estimation, (Smith and Eisner 2005)



A neural network for learning word vectors

How do we formalize this idea? Ask that

score(cat chills on a mat) > score(cat chills Jeju a mat)

How do we compute the score?

- With a neural network
- Each word is associated with an n -dimensional vector



Word embedding matrix

- Initialize all word vectors randomly to form a word embedding matrix $L \in \mathbb{R}^{n \times |V|}$

$$L = \begin{bmatrix} \bullet & \bullet & \bullet & \dots & \bullet & \bullet \\ \bullet & \bullet & \bullet & \dots & \bullet & \bullet \\ \bullet & \bullet & \bullet & \dots & \bullet & \bullet \\ \bullet & \bullet & \bullet & \dots & \bullet & \bullet \end{bmatrix}_n$$

the cat mat ...

- These are the word features we want to learn
- Also called a look-up table
 - Conceptually you get a word's vector by left multiplying a one-hot vector e by L : $x = Le$

Word vectors as input to a neural network

- $\text{score}(\text{cat chills on a mat})$
- To describe a phrase, retrieve (via index) the corresponding vectors from L



- Then concatenate them to $5n$ vector:
- $x = [\text{●●●●} \text{●●●●} \text{●●●●} \text{●●●●} \text{●●●●}]$
- How do we then compute $\text{score}(x)$?

The secret sauce is the unsupervised pre-training on a large text collection

(Collobert & Weston 2008; Collobert et al. 2011)



	POS WSJ (acc.)	NER CoNLL (F1)
State-of-the-art*	97.24	89.31
Supervised NN	96.37	81.47
Unsupervised pre-training followed by supervised NN**	97.20	88.87
+ hand-crafted features***	97.29	89.59

* Representative systems: POS: (Toutanova et al. 2003), NER: (Ando & Zhang 2005)

** 130,000-word embedding trained on Wikipedia and Reuters with 11 word window, 100 unit hidden layer – **for 7 weeks!** – then supervised task training

*** Features are character suffixes for POS and a gazetteer for NER

Supervised refinement of the unsupervised word representation helps

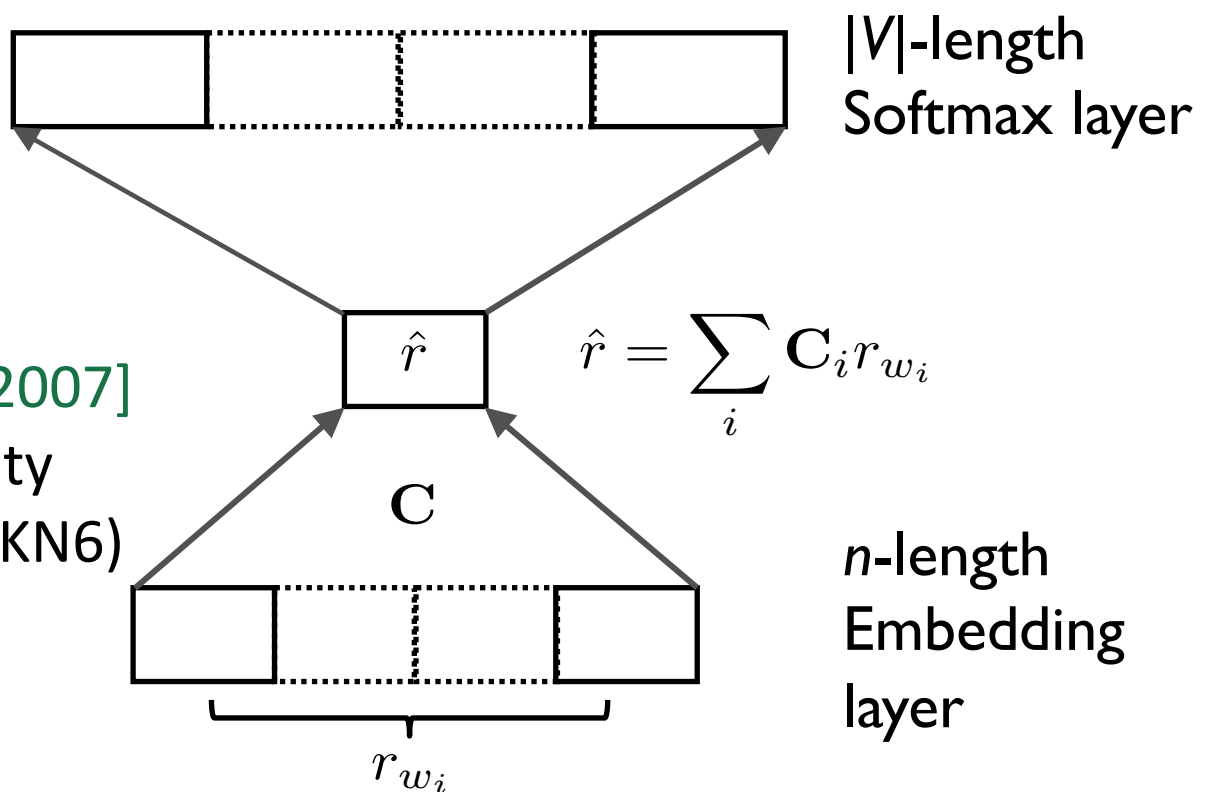
	POS WSJ (acc.)	NER CoNLL (F1)
Supervised NN	96.37	81.47
NN with Brown clusters	96.92	87.15
Fixed embeddings*	97.10	88.87
C&W 2011**	97.29	89.59

* Same architecture as C&W 2011, but word embeddings are kept constant during the supervised training phase

** C&W is unsupervised pre-train + supervised NN + features model of last slide

Bilinear Language Model

- Even a linear version of the Neural Language Model works better than n-grams



- [Mnih & Hinton 2007]
- APNews perplexity down from 117 (KN6) to 96.5

Language Modeling Output Bottleneck

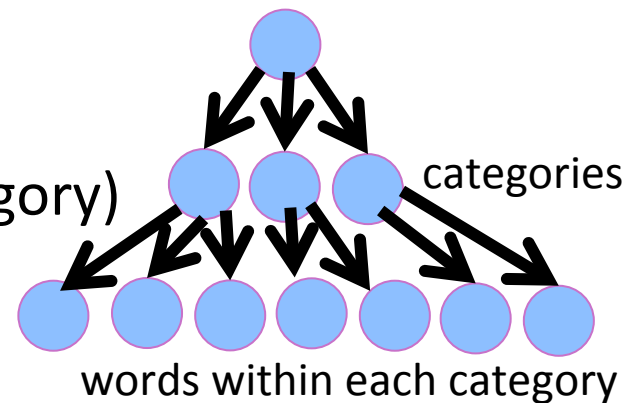
- [Schwenk et al 2002]: only predict most frequent words (short list) and use n-gram for the others



- [Morin & Bengio 2005; Blitzer et al 2005; Mnih & Hinton 2007,2009; Mikolov et al 2011]: **hierarchical representations**, multiple output groups, conditionally computed, predict

- $P(\text{word category} \mid \text{context})$
- $P(\text{sub-category} \mid \text{context, category})$
- $P(\text{word} \mid \text{context, sub-category, category})$

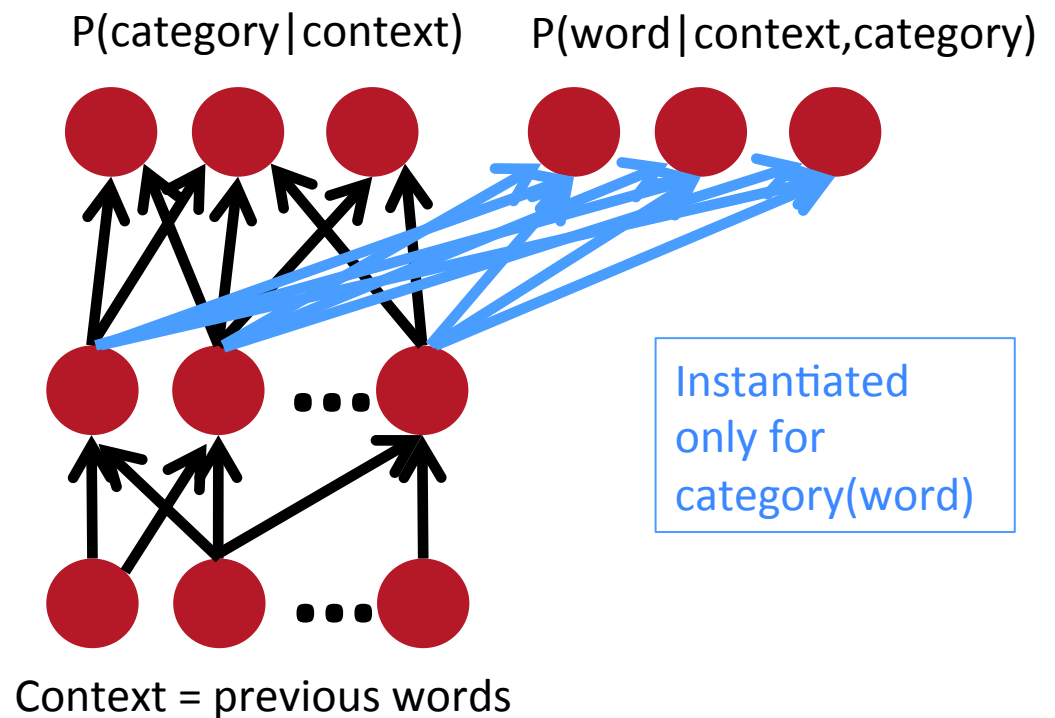
- Hard categories, can be arbitrary [Mikolov et al 2011]



Language Modeling Output Bottleneck: Hierarchical word categories

Compute

$P(\text{word} | \text{category}, \text{context})$
only for
 $\text{category} = \text{category}(\text{word})$



Language Modeling Output Bottleneck: Sampling Methods

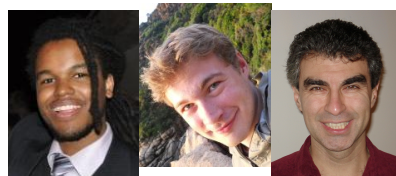
- Importance sampling to recover next-word probabilities [Bengio & Senecal 2003, 2008]



- Contrastive Sampling of negative examples, with a ranking loss [Collobert et al, 2008, 2011]
 - (no probabilities, ok if the goal is just to learn word embeddings)



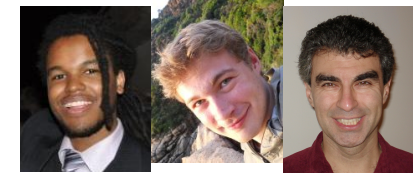
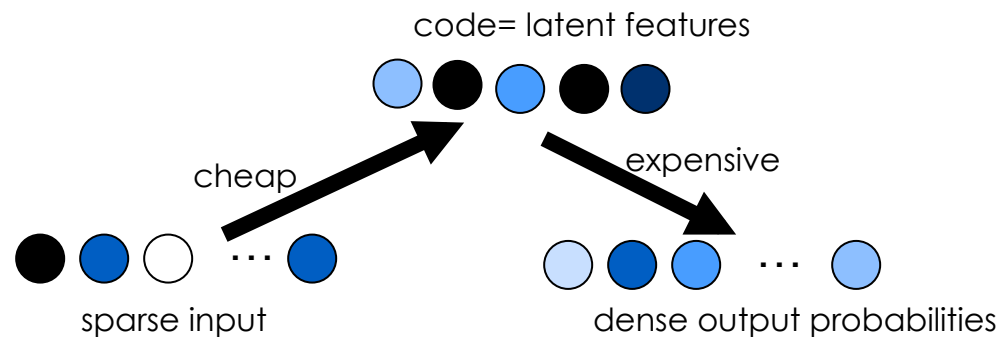
- Importance sampling for reconstructing bag-of-words [Dauphin et al 2011]



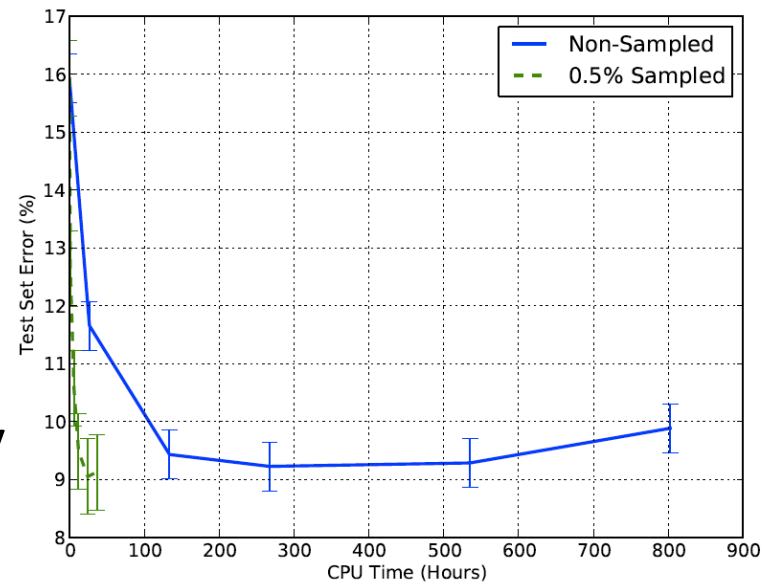
Sampled Reconstruction Trick

[Dauphin et al, ICML 2011]

- Auto-encoders and RBMs reconstruct the input, which is sparse and high-dimensional



- Applied to bag-of-words input for sentiment analysis, with denoising auto-encoders
- Always reconstruct the non-zeros in the input, and reconstruct as many randomly chosen zeros



Representing Sparse High-Dimensional Stuff: Sampled Reconstruction

$$L(\mathbf{x}, \mathbf{z}) = \sum_k^d H(\mathbf{x}_k, \mathbf{z}_k)$$

Stochastic reweighted loss

$$\hat{L}(\mathbf{x}, \mathbf{z}) = \sum_k \frac{\mathbf{p}_k}{\mathbf{q}_k} H(\mathbf{x}_k, \mathbf{z}_k)$$

Sample which inputs to reconstruct

$$\hat{\mathbf{p}} \in \{0, 1\}^d \text{ with } \hat{\mathbf{p}} \sim P(\hat{\mathbf{p}}|\mathbf{x})$$

$$\mathbf{q}_k = E[\hat{\mathbf{p}}_k | k, \mathbf{x}, \tilde{\mathbf{x}}]$$

Importance sampling reweighting

$$\text{Let } \mathcal{C}(\mathbf{x}, \tilde{\mathbf{x}}) = \{k : \mathbf{x}_k = 1 \text{ or } \tilde{\mathbf{x}}_k = 1\}$$

Minimum-variance: guess wrong reconstructions

$$P(\hat{\mathbf{p}}_k = 1 | \mathbf{x}_k) = \begin{cases} 1 & \text{if } k \in \mathcal{C}(\mathbf{x}, \tilde{\mathbf{x}}) \\ |\mathcal{C}(\mathbf{x}, \tilde{\mathbf{x}})|/d_x & \text{otherwise} \end{cases}$$

As many randomly chosen other bits

Recurrent Neural Net Language Modeling for ASR

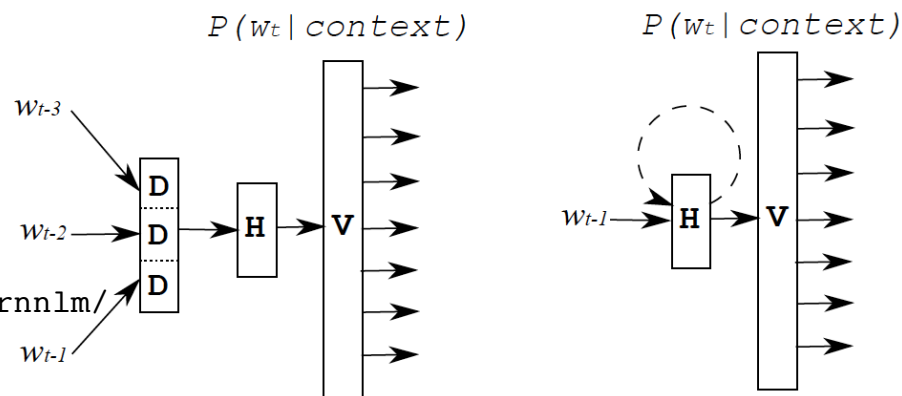
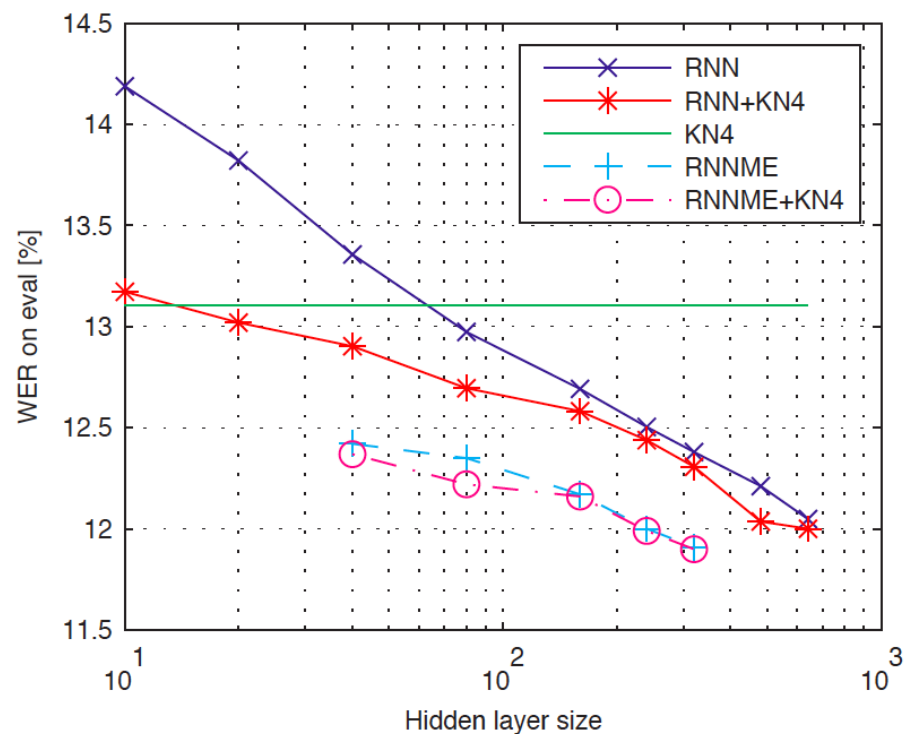
- [Mikolov et al 2011]
Bigger is better...
experiments on Broadcast
News NIST-RT04



perplexity goes from
140 to 102

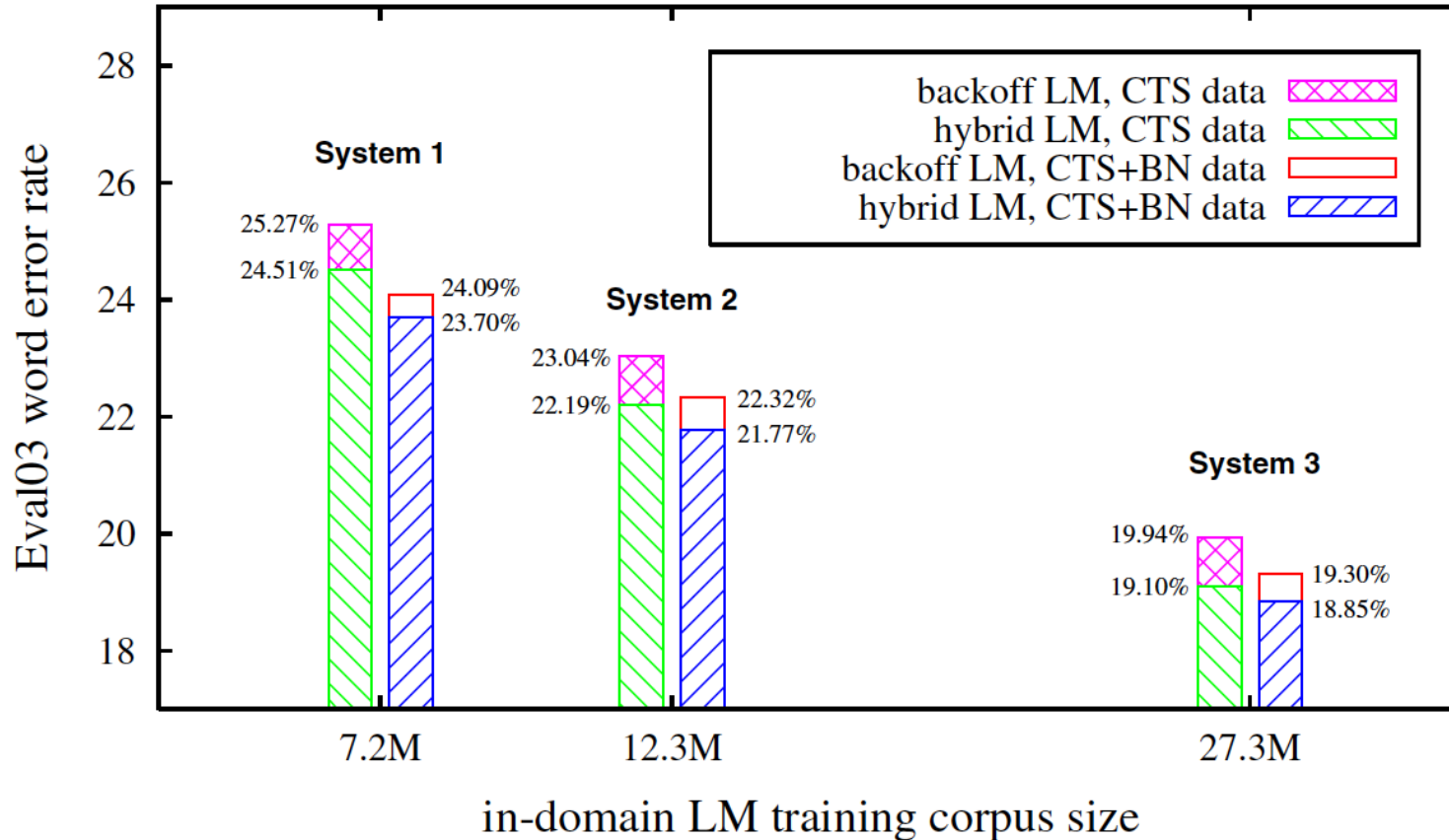
Paper shows how to
train a recurrent neural net
with a single core in a few
days, with > 1% absolute
improvement in WER

Code: <http://www.fit.vutbr.cz/~imikolov/rnnlm/>



Neural Net Language Modeling for ASR

- [Schwenk 2007], real-time ASR, perplexity AND word error rate improve (CTS evaluation set 2003), perplexities go from 50.1 to 45.5



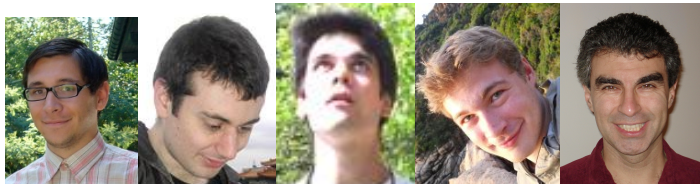
Application to Statistical Machine Translation



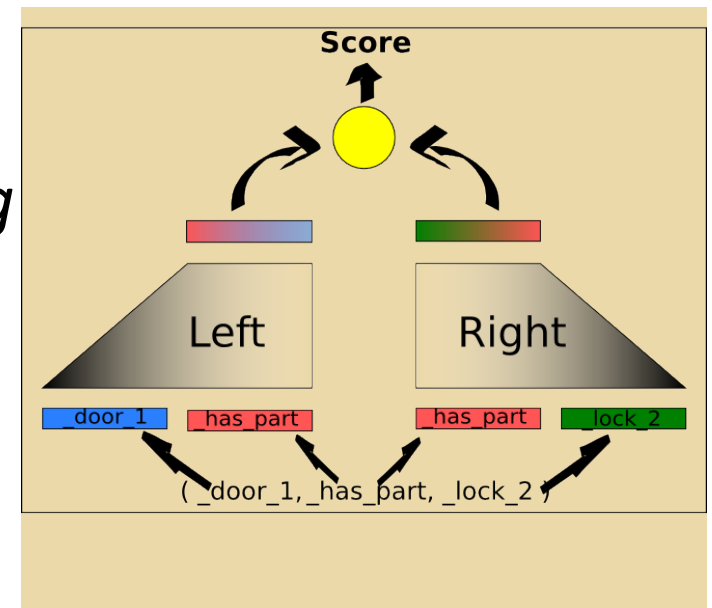
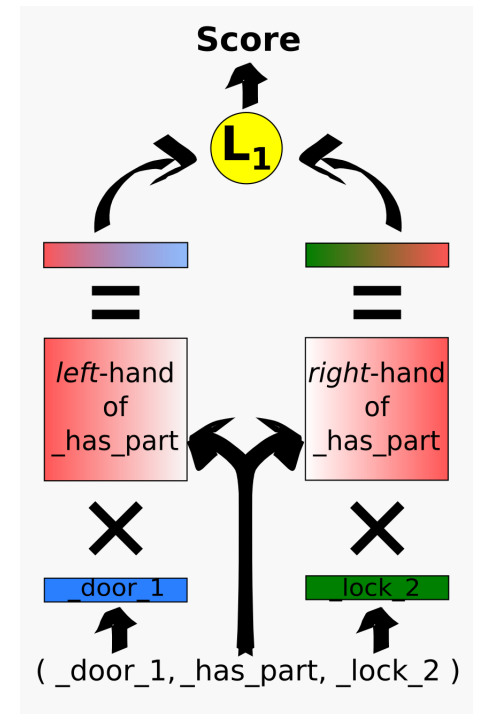
- Schwenk (NAACL 2012 workshop on the future of LM)
 - 41M words, Arabic/English bitexts + 151M English from LDC
- Perplexity down from 71.1 (6 Gig back-off) to 56.9 (neural model, 500M memory)
- +1.8 BLEU score (50.75 to 52.28)
- Can take advantage of longer contexts
- Code: <http://lium.univ-lemans.fr/cs1m/>

Modeling Semantics

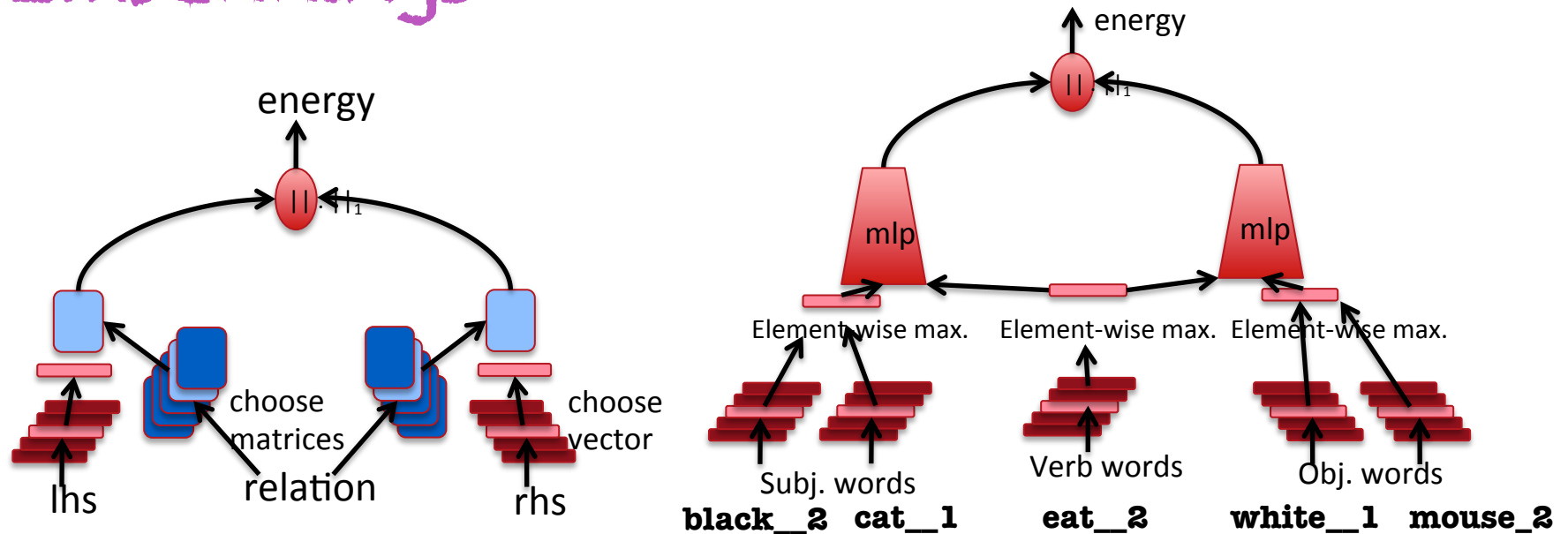
Learning Structured Embeddings of Knowledge Bases, (Bordes, Weston, Collobert & Bengio, AAAI 2011)



Joint Learning of Words and Meaning Representations for Open-Text Semantic Parsing, (Bordes, Glorot, Weston & Bengio, AISTATS 2012)



Modeling Relations: Operating on Embeddings



Model (lhs, relation, rhs)

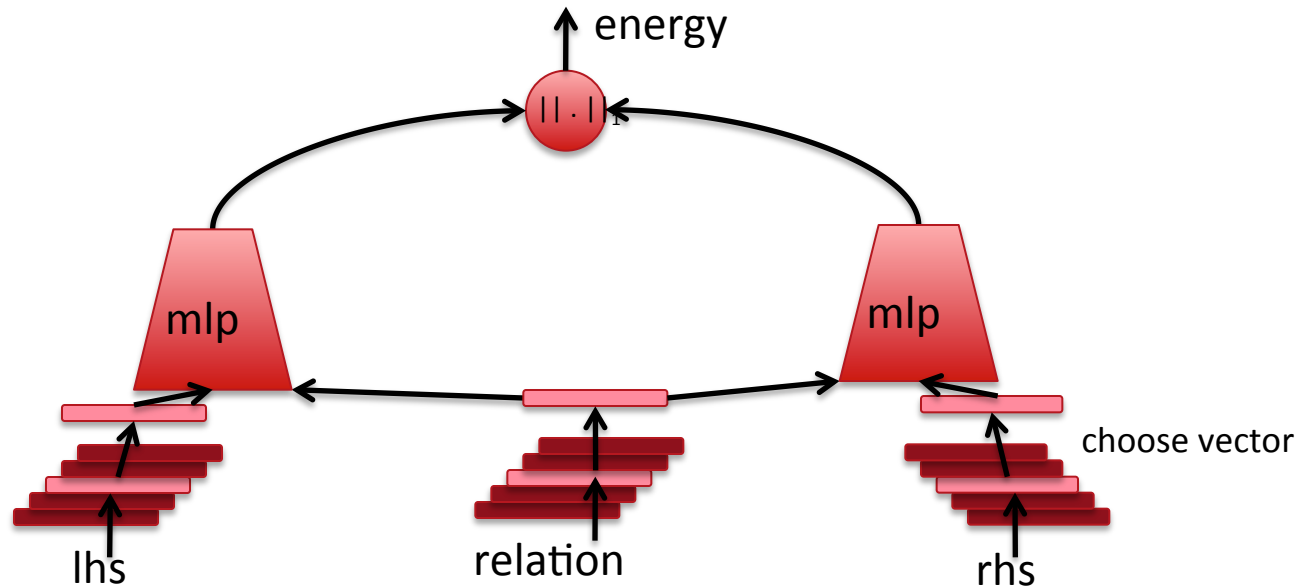
Each concept = 1 embedding vector

Each relation = 2 matrices. **Matrix or mlp acts as operator.**

Ranking criterion

Energy = low for training examples, high o/w

Allowing Relations on Relations



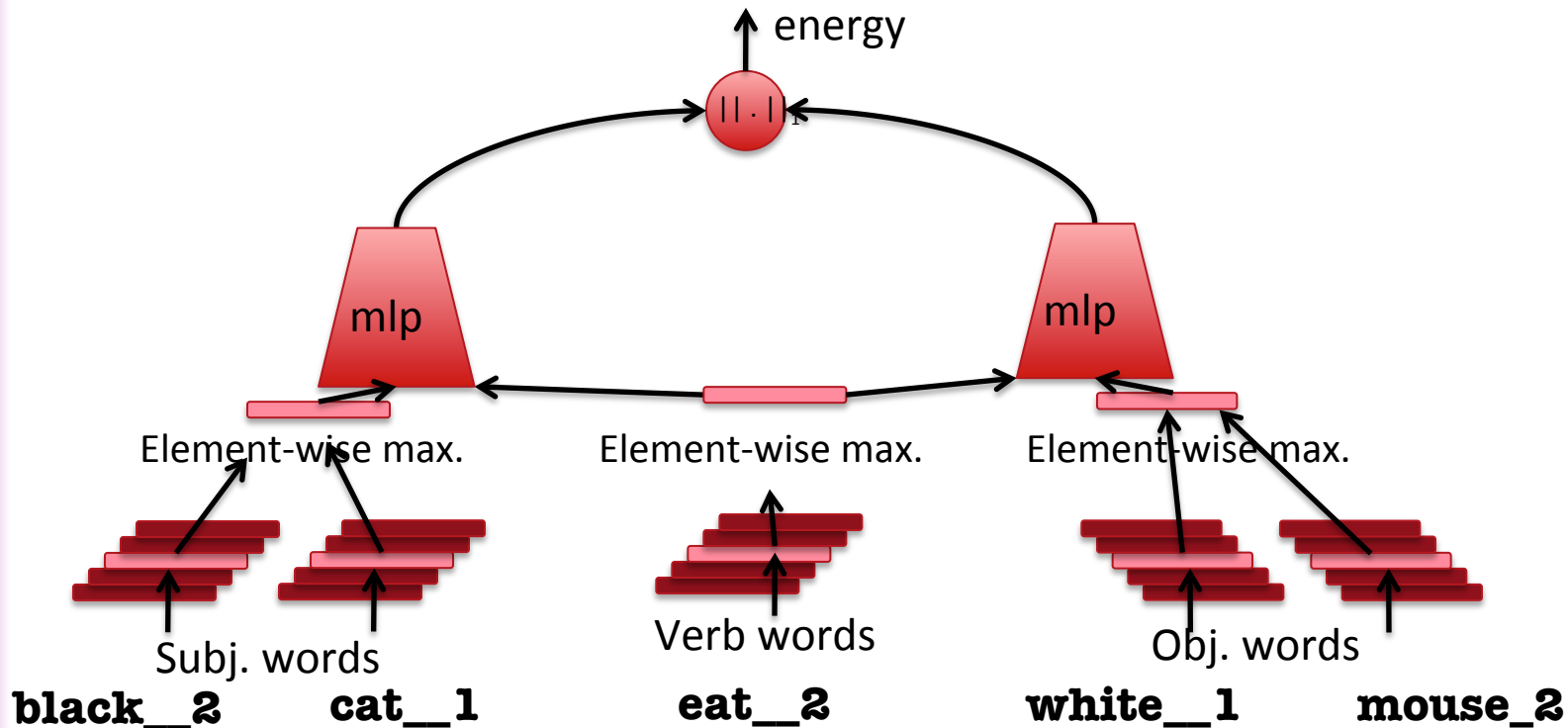
Verb = relation. Too many to have a matrix each.

Each concept = 1 embedding vector

Each relation = 1 embedding vector

Can handle **relations on relations on relations**

Training on Full Sentences



Use SENNA (Collobert et al 2011) = embedding-based NLP tagger for Semantic Role Labeling, breaks sentence into (subject, verb, object) phrases

→ Use max-pooling to aggregate embeddings of words inside each part

Open-Text Semantic Parsing

- 3 steps:

``A musical score accompanies a television program ."

↓ **Semantic Role Labeling**

(``A musical score", ``accompanies", ``a television program")

↓ **Preprocessing (POS, Chunking, ...)**

((`_musical_JJ` `score_NN`), `_accompany_VB` , `_television_program_NN`)

↓ **Word-sense Disambiguation**

((`_musical_JJ_1` `score_NN_2`), `_accompany_VB_1`, `_television_program_NN_1`)

- last formula defines the Meaning Representation (MR).

Training Criterion

- Intuition: if an entity of a triplet was missing, we would like our model to predict it correctly i.e. to give it the lowest energy. For example, this would allow us to answer questions like “what is part of a car?”
- Hence, for any training triplet $x_i = (lhs_i, rel_i, rhs_i)$ we would like:
 - (1) $E(lhs_i, rel_i, rhs_i) < E(lhs_j, rel_i, rhs_i),$
 - (2) $E(lhs_i, rel_i, rhs_i) < E(lhs_i, rel_j, rhs_i),$
 - (3) $E(lhs_i, rel_i, rhs_i) < E(lhs_i, rel_i, rhs_j),$

That is, the energy function E is trained to rank training samples below all other triplets.

Contrastive Sampling of Neg. Ex. = pseudo-likelihood + uniform sampling of negative variants

Train by stochastic gradient descent:

1. Randomly select a **positive training triplet** $x_i = (\text{lhs}_i, \text{rel}_i, \text{rhs}_i)$.
2. Randomly select constraint (1), (2) or (3) and an entity \tilde{e} :
 - If constraint (1), construct **negative triplet** $\tilde{x} = (\tilde{e}, \text{rel}_i, \text{rhs}_i)$.
 - Else if constraint (2), construct $\tilde{x} = (\text{lhs}_i, \tilde{e}, \text{rhs}_i)$.
 - Else, construct $\tilde{x} = (\text{lhs}_i, \text{rel}_i, \tilde{e})$.
3. If $E(x_i) > E(\tilde{x}) - 1$ make a **gradient step** to minimize:
 $\max(0, 1 - E(\tilde{x}) + E(x_i))$.
4. Constraint embedding vectors to norm 1

Question Answering: implicitly adding new relations to WN or FB

	Model (All)	<i>TextRunner</i>
<i>lhs</i>	_army_NN_1	<i>army</i>
<i>rel</i>	_attack_VB_1	<i>attacked</i>
top ranked <i>rhs</i>	_troop_NN_4 _armed_service_NN_1 _ship_NN_1 _territory_NN_1 _military_unit_NN_1	<i>Israel</i> <i>the village</i> <i>another army</i> <i>the city</i> <i>the fort</i>
top ranked <i>lhs</i>	_business_firm_NN_1 _person_NN_1 _family_NN_1 _payoff_NN_3 _card_game_NN_1	<i>People</i> <i>Players</i> <i>one</i> <i>Students</i> <i>business</i>
<i>rel</i>	_earn_VB_1	<i>earn</i>
<i>rhs</i>	_money_NN_1	<i>money</i>

MRs inferred from text define triplets between WordNet synsets.

Model captures knowledge about relations between nouns and verbs.

→ Implicit addition of new relations to WordNet!

→ Generalize Freebase!

Embedding Nearest Neighbors of Words & Senses

<p>_mark_NN</p> <p>_indication_NN _print_NN_3 _print_NN _roll_NN _pointer_NN</p>	<p>_mark_NN_1</p> <p>_score_NN_1 _number_NN_2 _gradation_NN _evaluation_NN_1 _tier_NN_1</p>	<p>_mark_NN_2</p> <p>_marking_NN_1 _symbolizing_NN_1 _naming_NN_1 _marking_NN _punctuation_NN_3</p>
<p>_take_VB</p> <p>_bring_VB _put_VB _ask_VB _hold_VB _provide_VB</p>	<p>_canary_NN</p> <p>_sea_mew_NN_1 _yellowbird_NN_2 _canary_bird_NN_1 _larus_marinus_NN_1 _mew_NN</p>	<p>_different_JJ_1</p> <p>_eccentric_NN _dissimilar_JJ _same_JJ_2 _similarity_NN_1 _common_JJ_1</p>

Word Sense Disambiguation

- Senseval-3 results (only sentences with Subject-Verb-Object structure)

MFS=most frequent sense

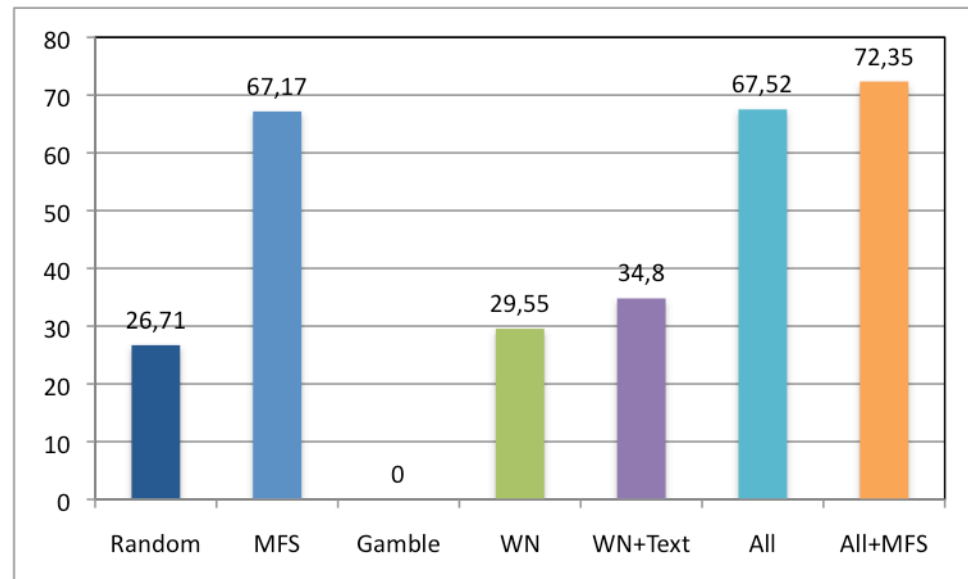
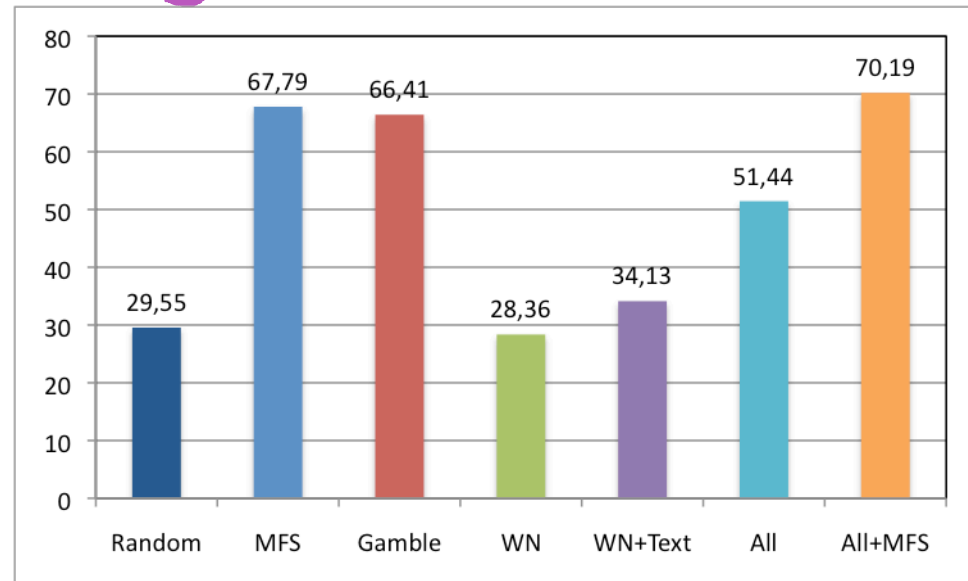
All=training from all sources

Gamble=Decadt et al 2004

(Senseval-3 SOA)

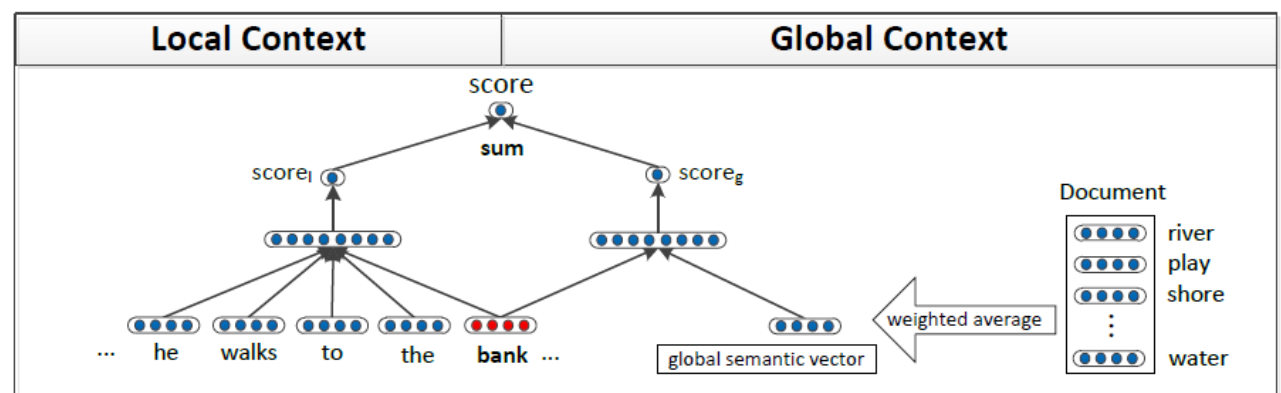
- XWN results

XWN = eXtended WN



Learning Multiple Word Vectors

- Tackles problems with polysemous words
- Can be done with both standard tf-idf based methods [Reisinger and Mooney, NAACL 2010]
- Recent neural word vector model by [Huang et al. ACL 2012] learns multiple prototypes using both local and global context
- State of the art correlations with human similarity judgments



Learning Multiple Word Vectors

- Visualization of learned word vectors from Huang et al. (ACL 2012)



Phoneme-Level Acoustic Models

- [Mohamed et al, 2011, IEEE Tr.ASLP]

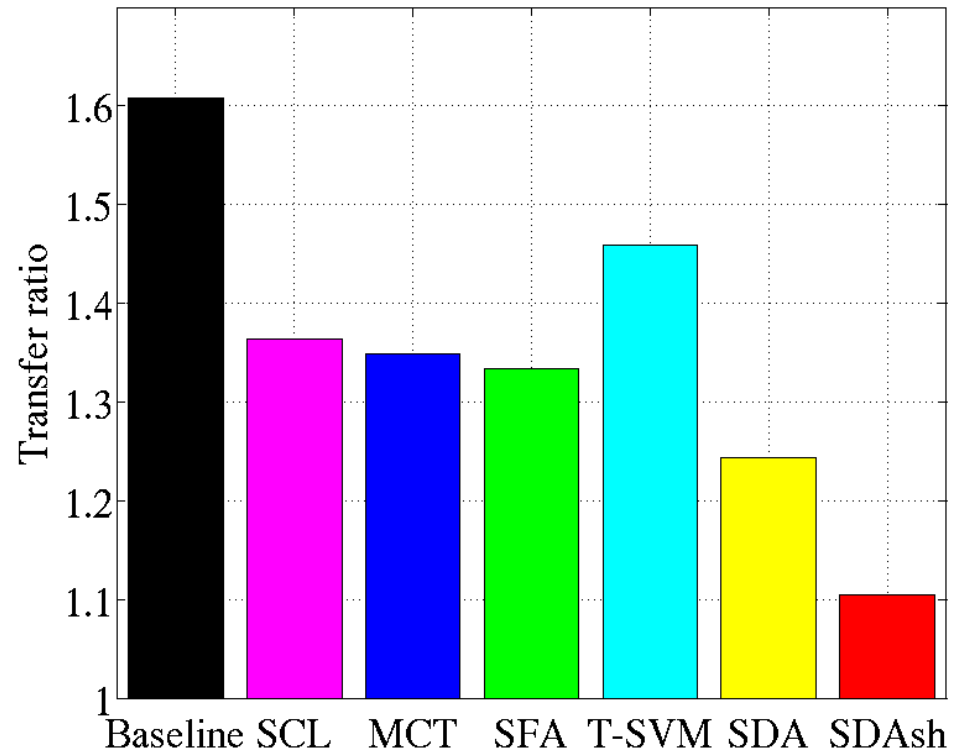


- Unsupervised pre-training as Deep Belief Nets (a stack of RBMs), supervised fine-tuning to predict phonemes
- Phoneme classification on TIMIT:
 - CD-HMM: 27.3% error
 - CRFs: 26.6%
 - Triphone HMMs w. BMML: 22.7%
 - Unsupervised DBNs: 24.5%
 - Fine-tuned DBNs: 20.7%
- Improved version by Dong Yu is **RELEASED IN MICROSOFT'S ASR** system for Audio Video Indexing Service

Domain Adaptation for Sentiment Analysis

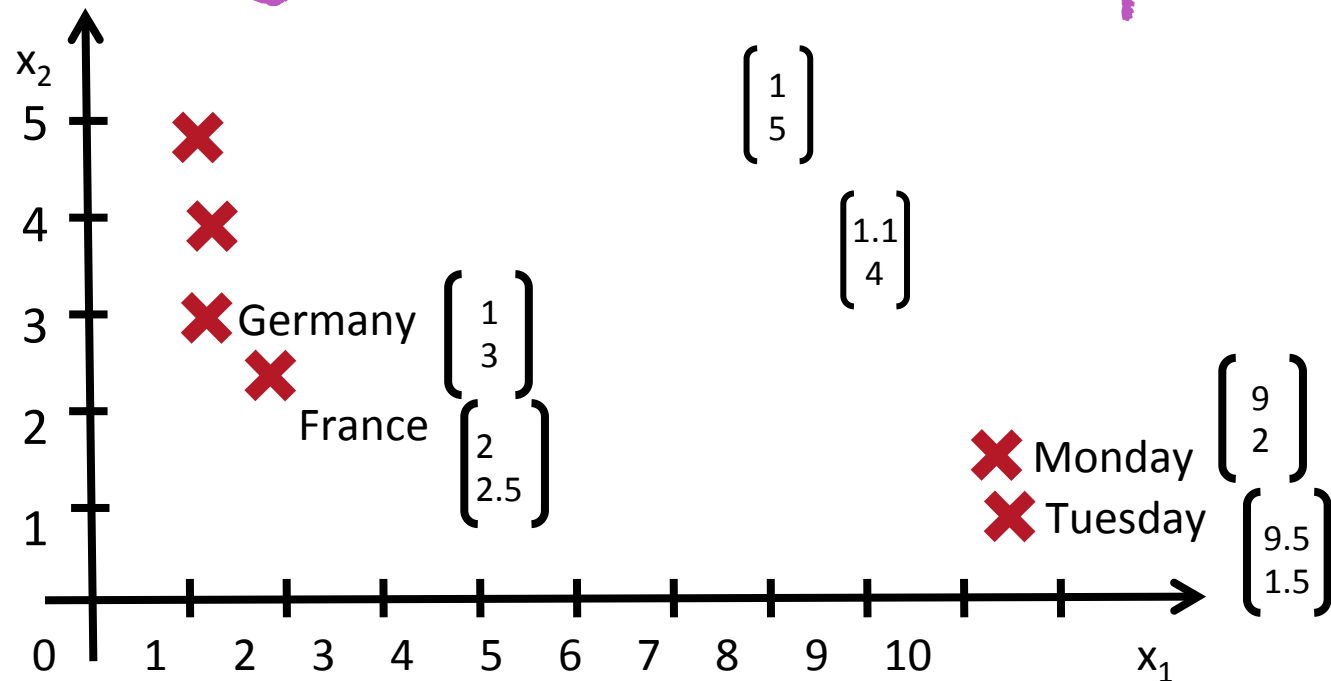


- [Glorot et al, ICML 2011] beats SOTA on Amazon benchmark, 25 domains
- Embeddings pre-trained in denoising auto-encoder
- Disentangling effect (features specialize to domain or sentiment)



Recursive Neural Networks

Building on Word Vector Space Models



the country of my birth
the place where I was born

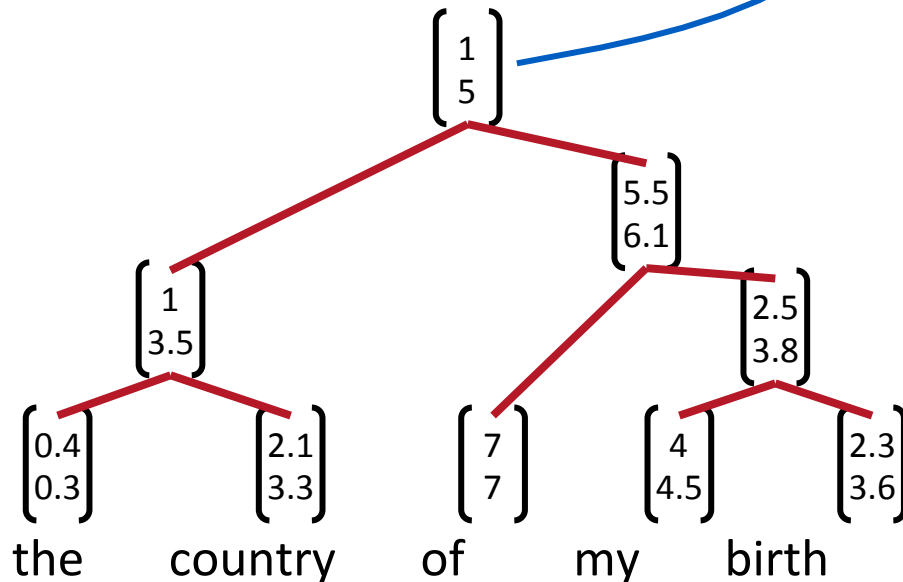
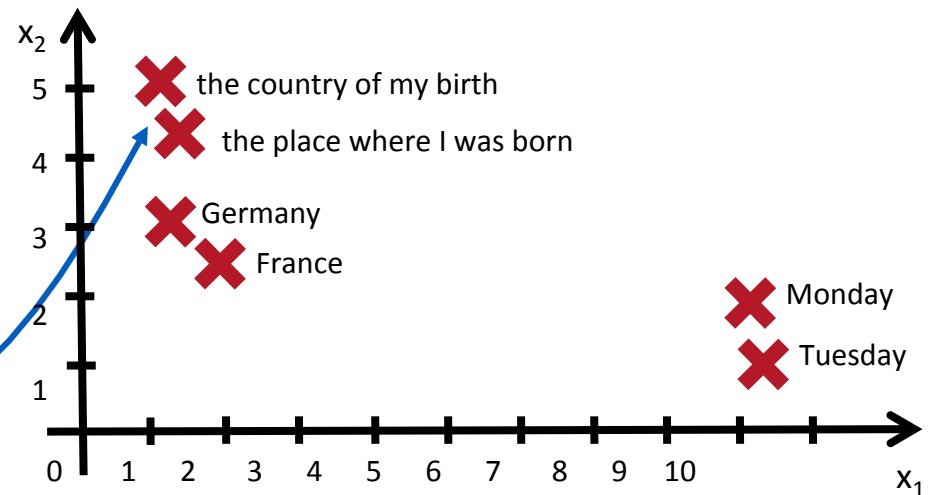
But how can we represent the meaning of longer phrases?

By mapping them into the same vector space!

How should we map phrases into a vector space?

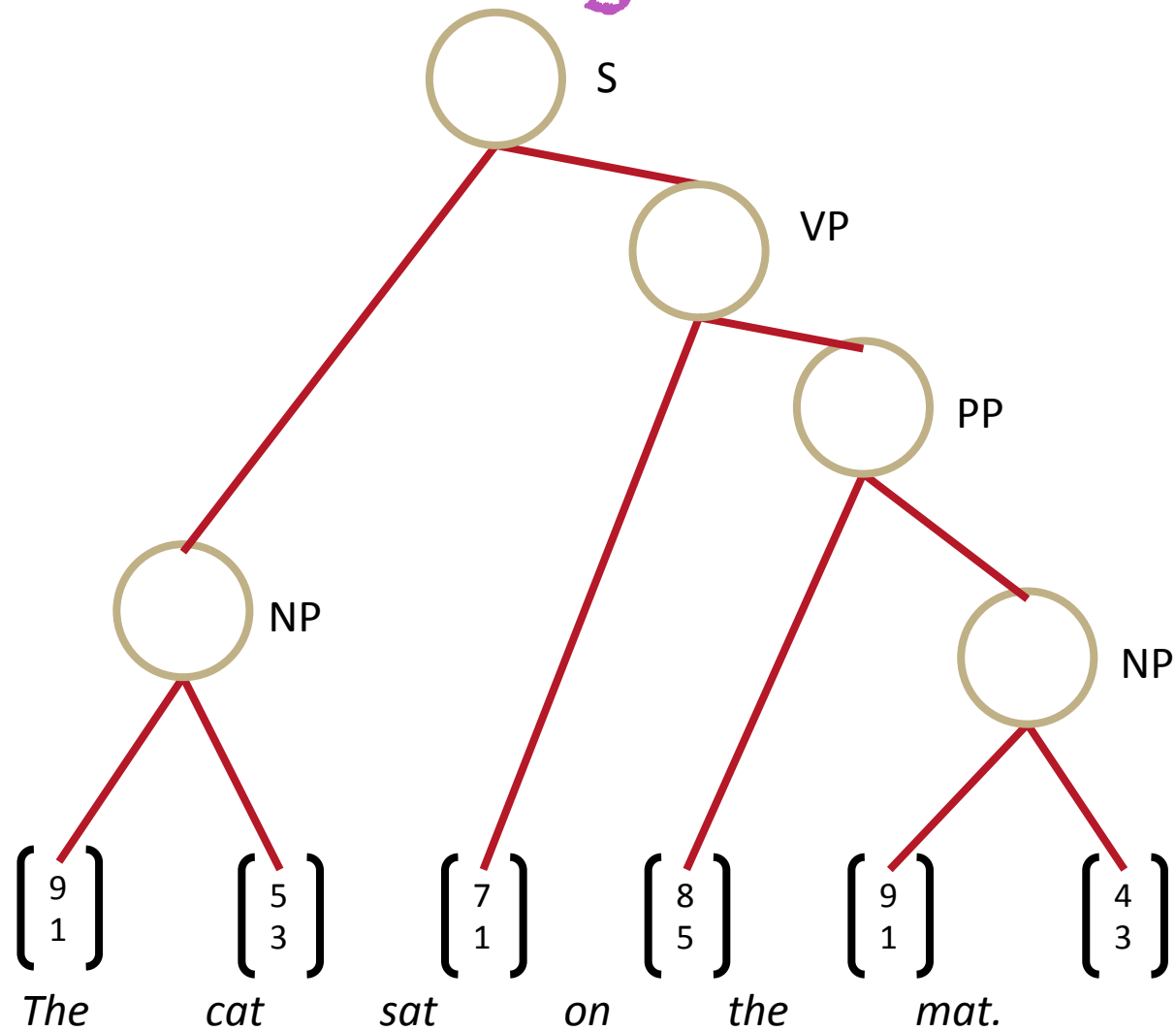
Use principle of compositionality

The meaning (vector) of a sentence is determined by
(1) the meanings of its words and
(2) the rules that combine them.

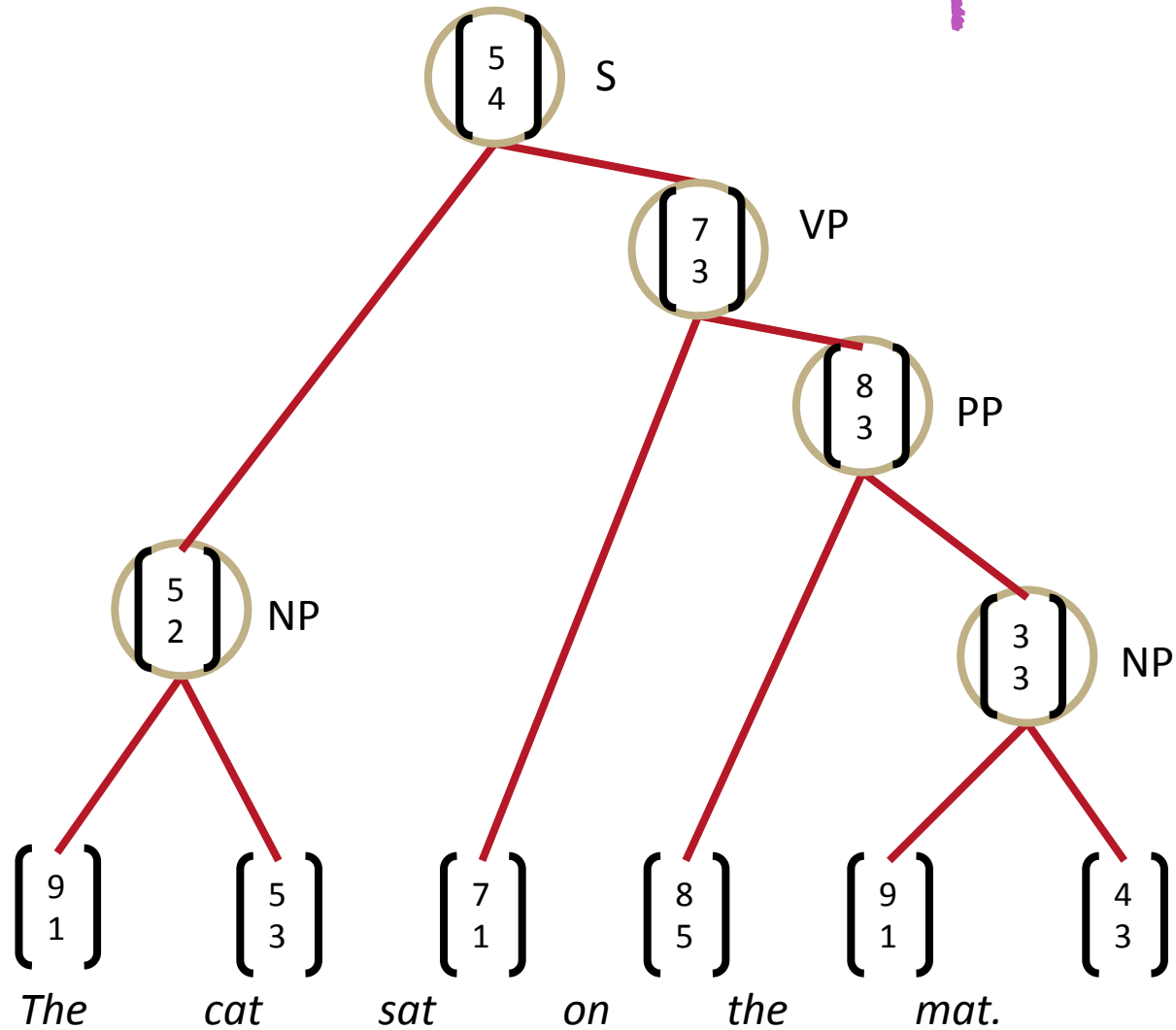


Recursive Neural Nets
can jointly learn
compositional vector
representations and
parse trees

Sentence Parsing: What we want



Learn Structure and Representation

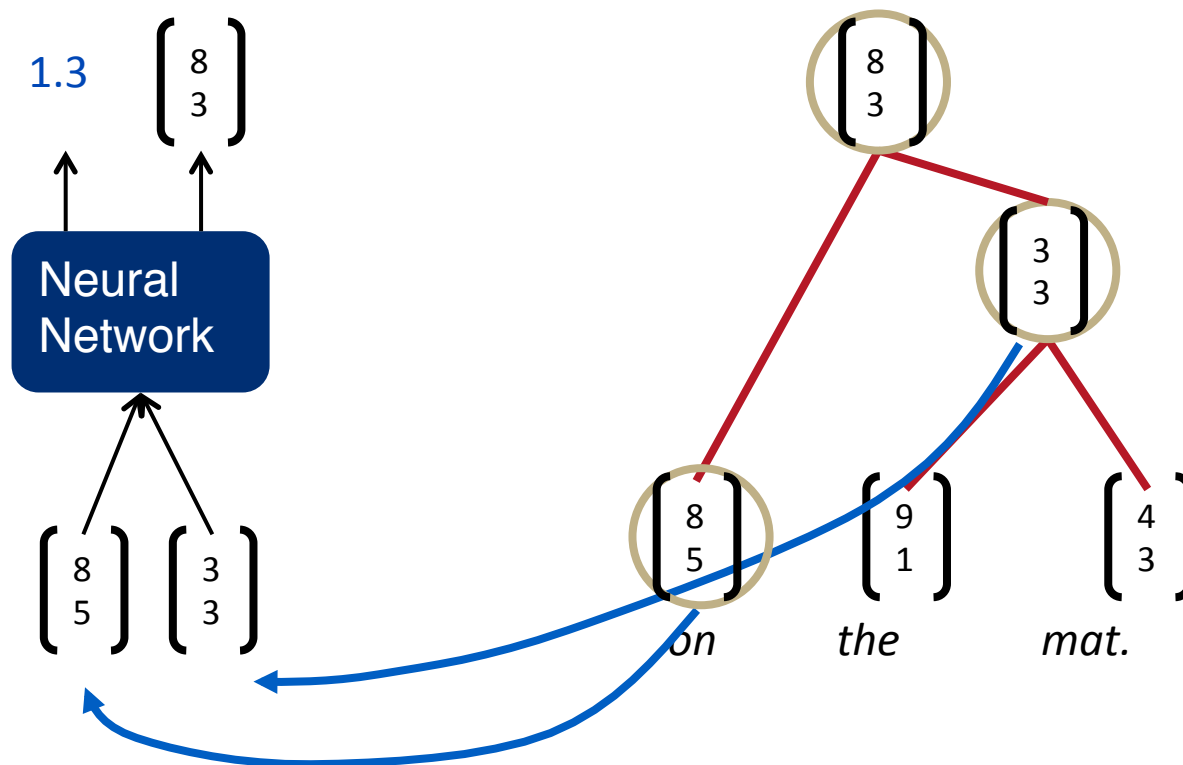


Recursive Neural Networks for Structure Prediction

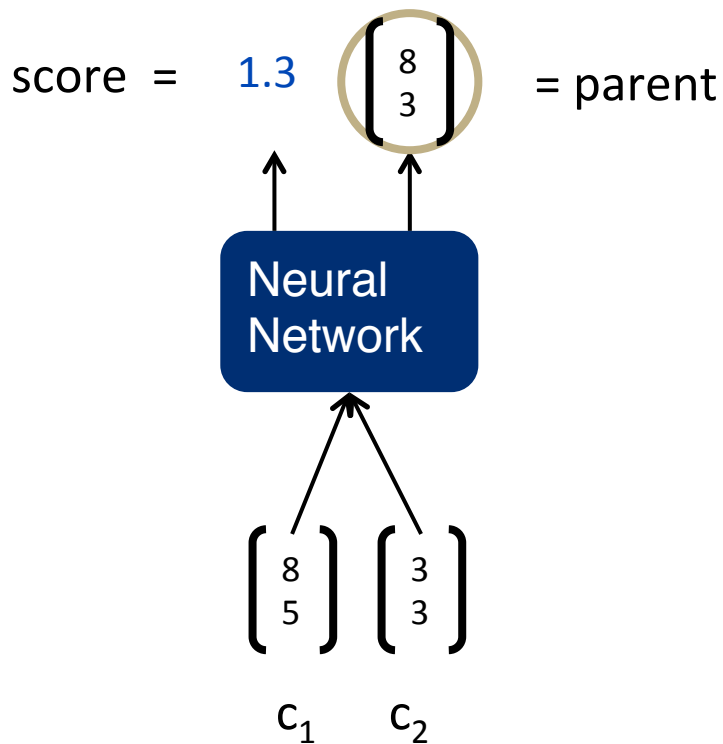
Inputs: two candidate children's representations

Outputs:

1. The semantic representation if the two nodes are merged.
2. Score of how plausible the new node would be.



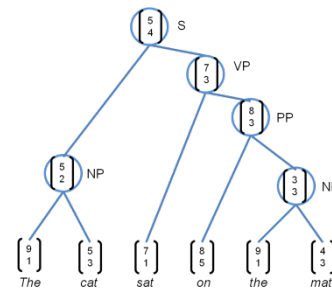
Recursive Neural Network Definition



$$\text{score} = U^T p$$

$$p = \tanh \left(W \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} + b \right)$$

Same W parameters at all nodes of the tree



Related Work to Socher et al. (ICML 2011)

- Pollack (1990): Recursive auto-associative memories



- Previous Recursive Neural Networks work by Goller & Küchler (1996), Costa et al. (2003) assumed fixed tree structure and used one hot vectors.

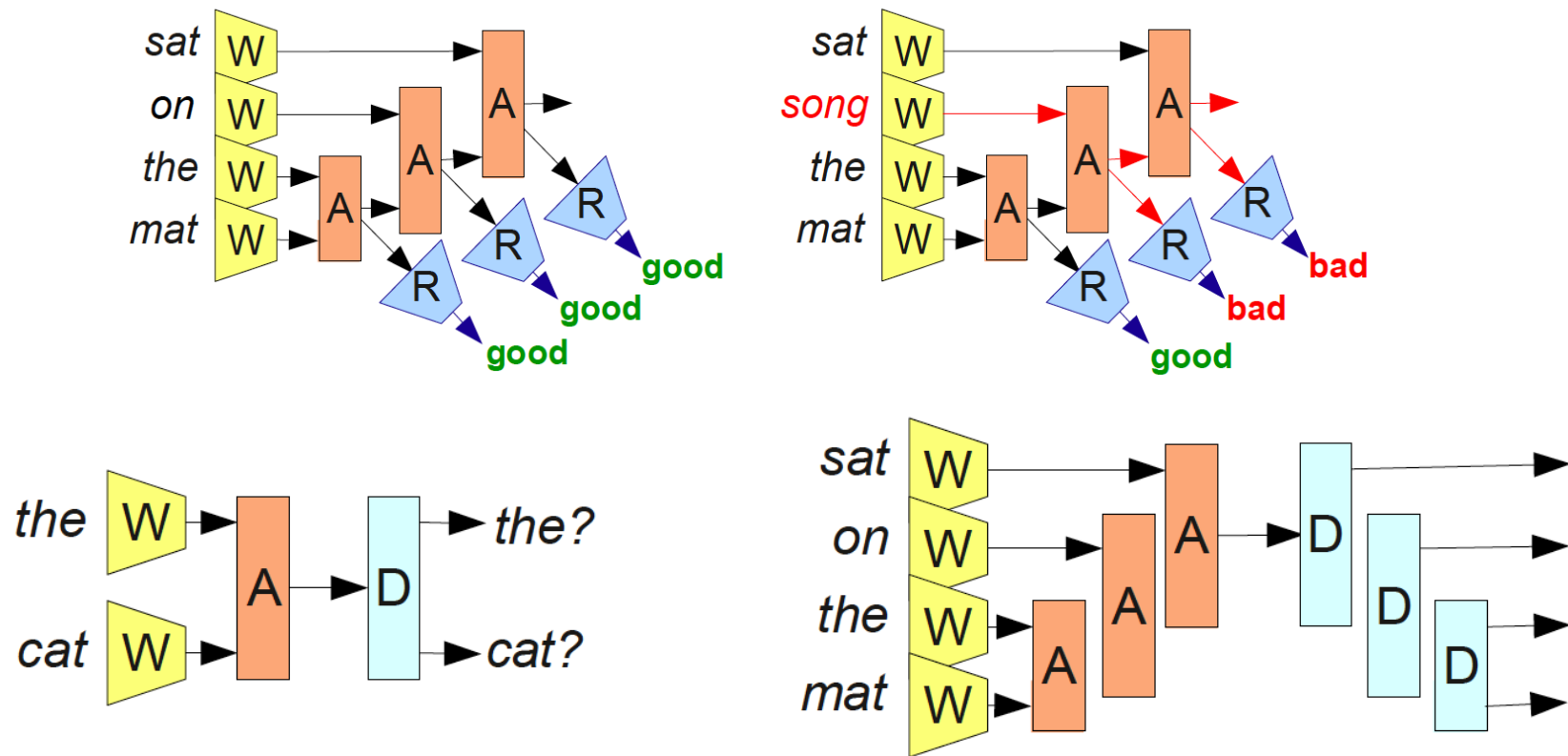


- Hinton (1990) and Bottou (2011): Related ideas about recursive models and recursive operators as smooth versions of logic operations

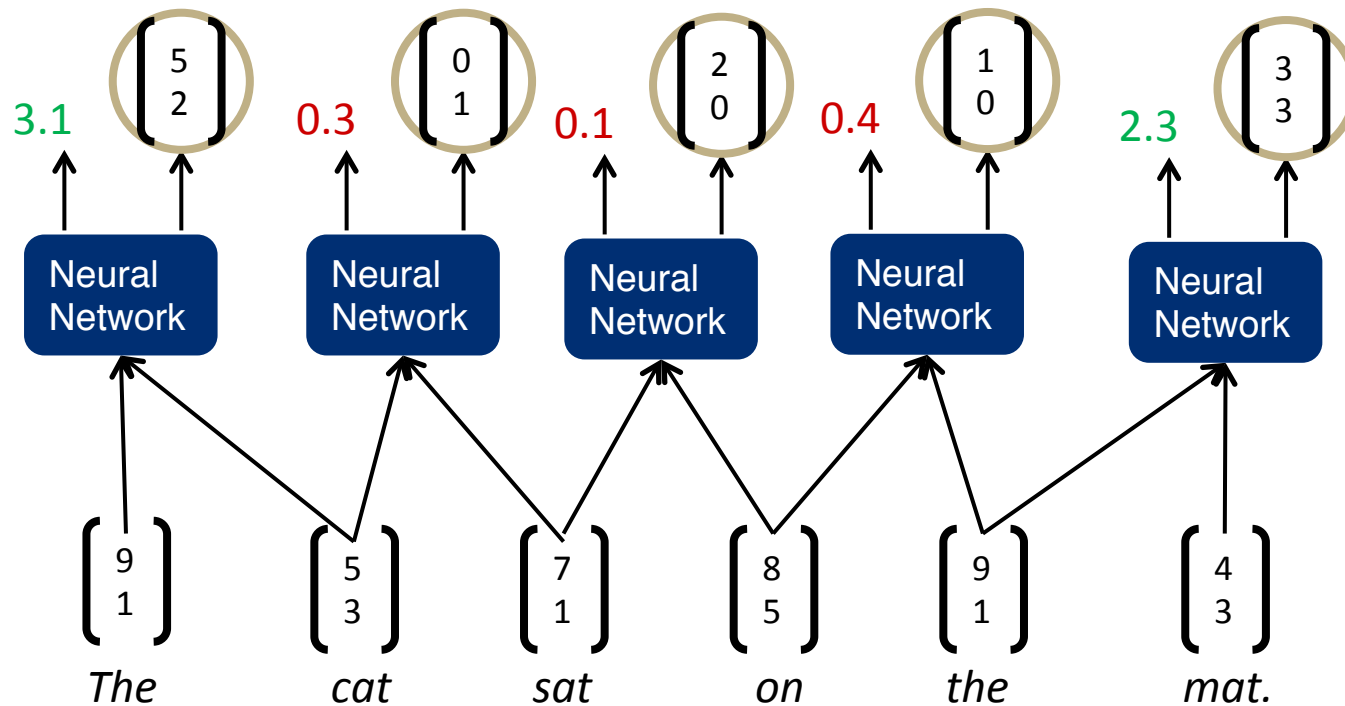


Recursive Application of Relational Operators

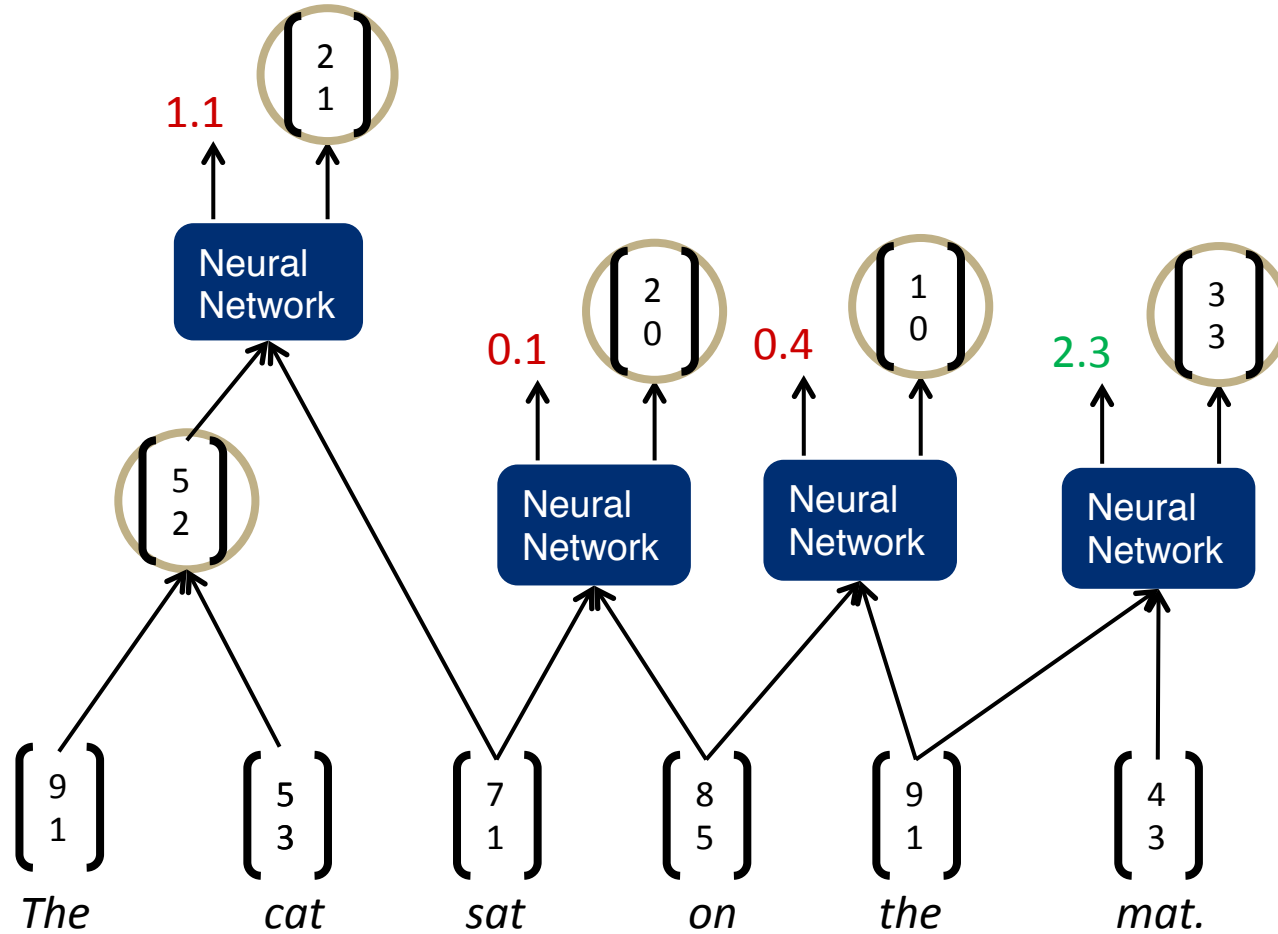
Bottou 2011: 'From machine learning to machine reasoning', also Socher ICML2011.



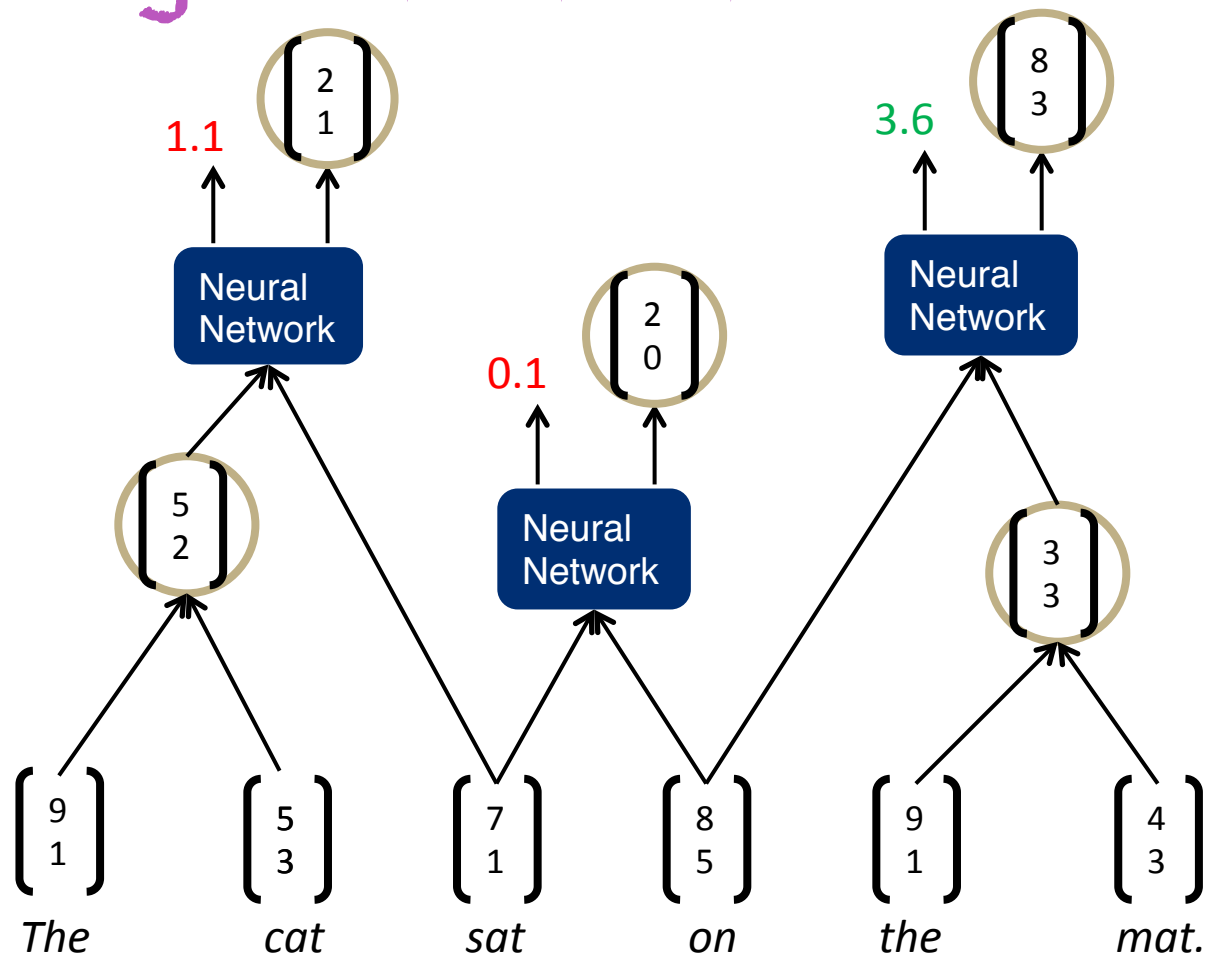
Parsing a sentence with an RNN



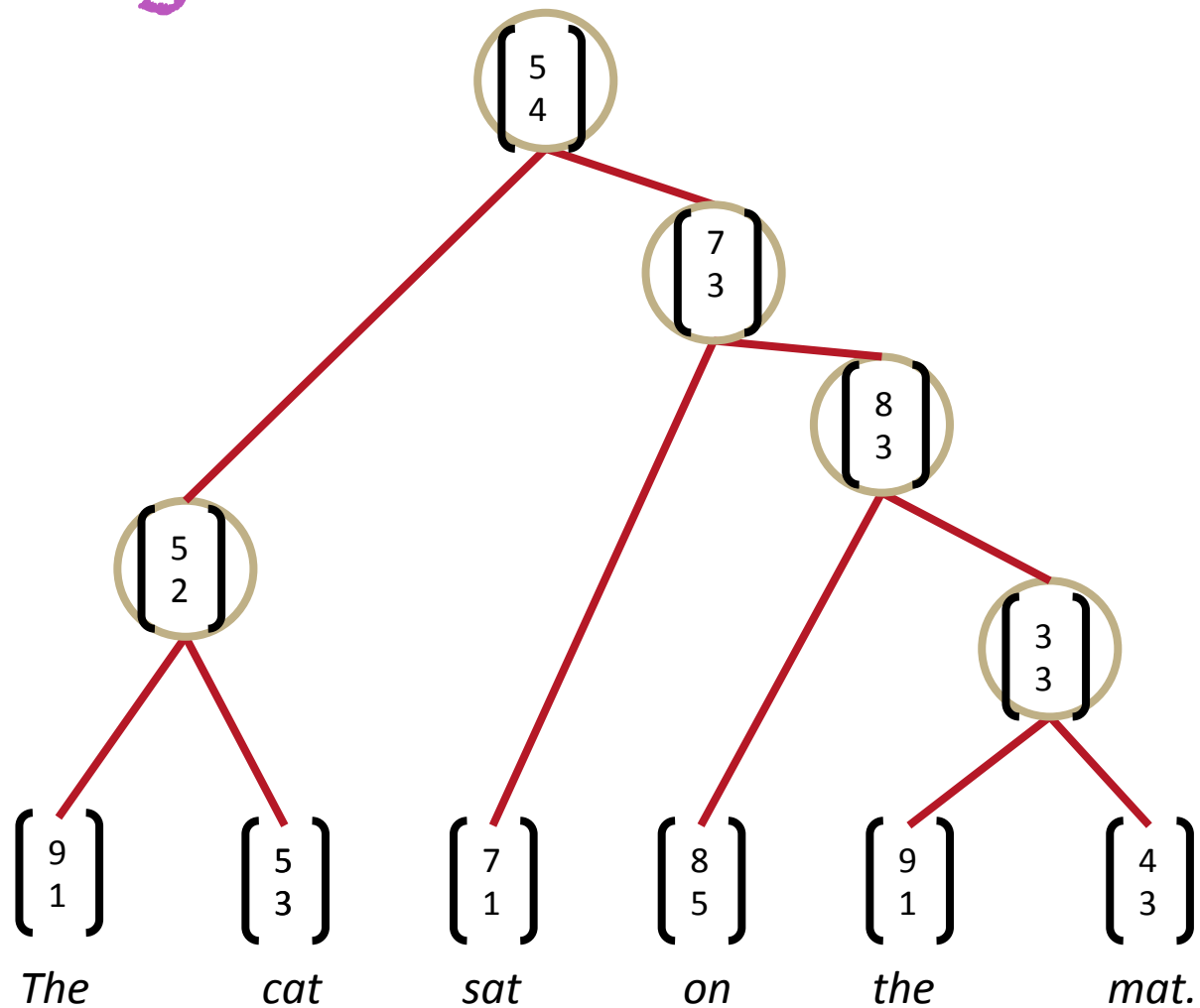
Parsing a sentence



Parsing a sentence



Parsing a sentence



Max-Margin Framework - Details

- The score of a tree is computed by the sum of the parsing decision scores at each node.



- Similar to max-margin parsing (Taskar et al. 2004), a supervised max-margin objective

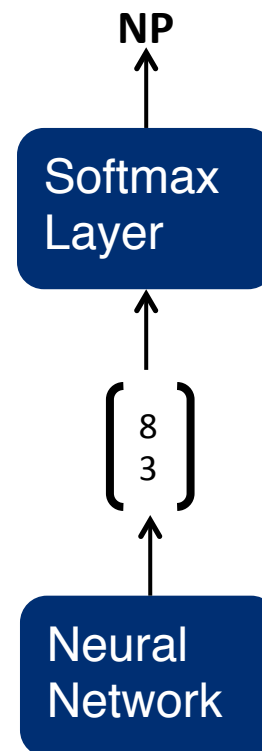
$$J = \sum_i s(x_i, y_i) - \max_{y \in A(x_i)} (s(x_i, y) + \Delta(y, y_i))$$

- The loss $\Delta(y, y_i)$ penalizes all incorrect decisions

Labeling in Recursive Neural Networks

- We can use each node's representation as features for a *softmax* classifier:

$$p(c|p) = \textit{softmax}(Sp)$$



Experiments: Parsing Short Sentences

- Standard *WSJ* train/test
- Good results on short sentences
- More work is needed for longer sentences

Model	L15 Dev	L15 Test
Recursive Neural Network	92.1	90.3
Sigmoid NN (Titov & Henderson 2007)	89.5	89.3
Berkeley Parser (Petrov & Klein 2006)	92.1	91.6

All the figures are adjusted for seasonal variations

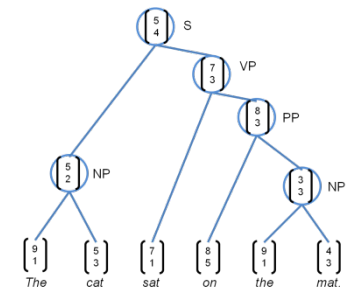
1. All the numbers are adjusted for seasonal fluctuations
2. All the figures are adjusted to remove usual seasonal patterns

Knight-Ridder wouldn't comment on the offer

1. Harsco declined to say what country placed the order
2. Coastal wouldn't disclose the terms

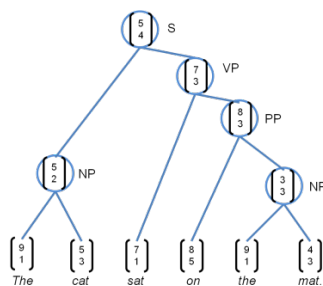
Sales grew almost 7% to \$UNK m. from \$UNK m.

1. Sales rose more than 7% to \$94.9 m. from \$88.3 m.
2. Sales surged 40% to UNK b. yen from UNK b.

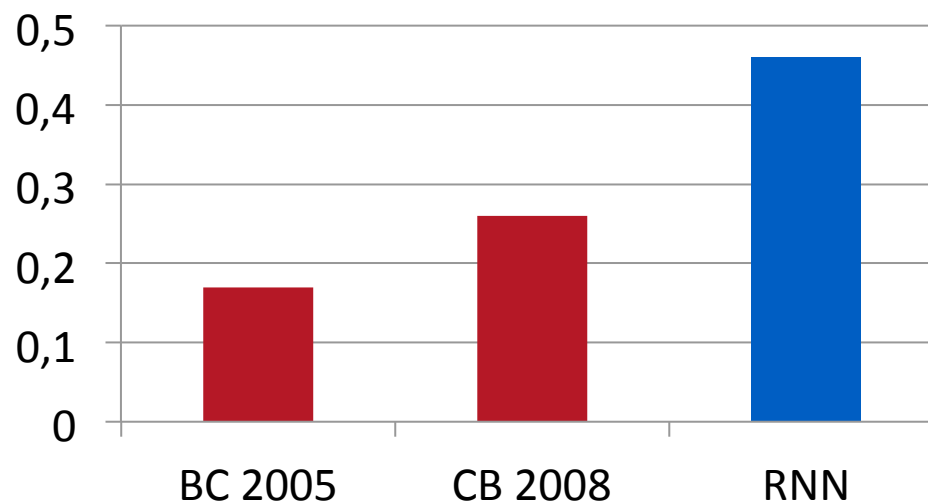


Short Paraphrase Detection

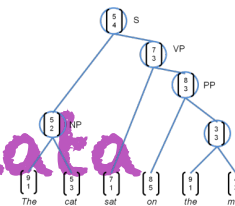
- Goal is to say which of candidate phrases are a good paraphrase of a given phrase
 - Motivated by Machine Translation
 - Initial algorithms: **Bannard & Callison-Burch 2005** (BC 2005), **Callison-Burch 2008** (CB 2008) exploit bilingual sentence-aligned corpora and hand-built linguistic constraints
 - Re-use system trained on parsing the WSJ



F1 of Paraphrase Detection



Paraphrase detection task, CCB data

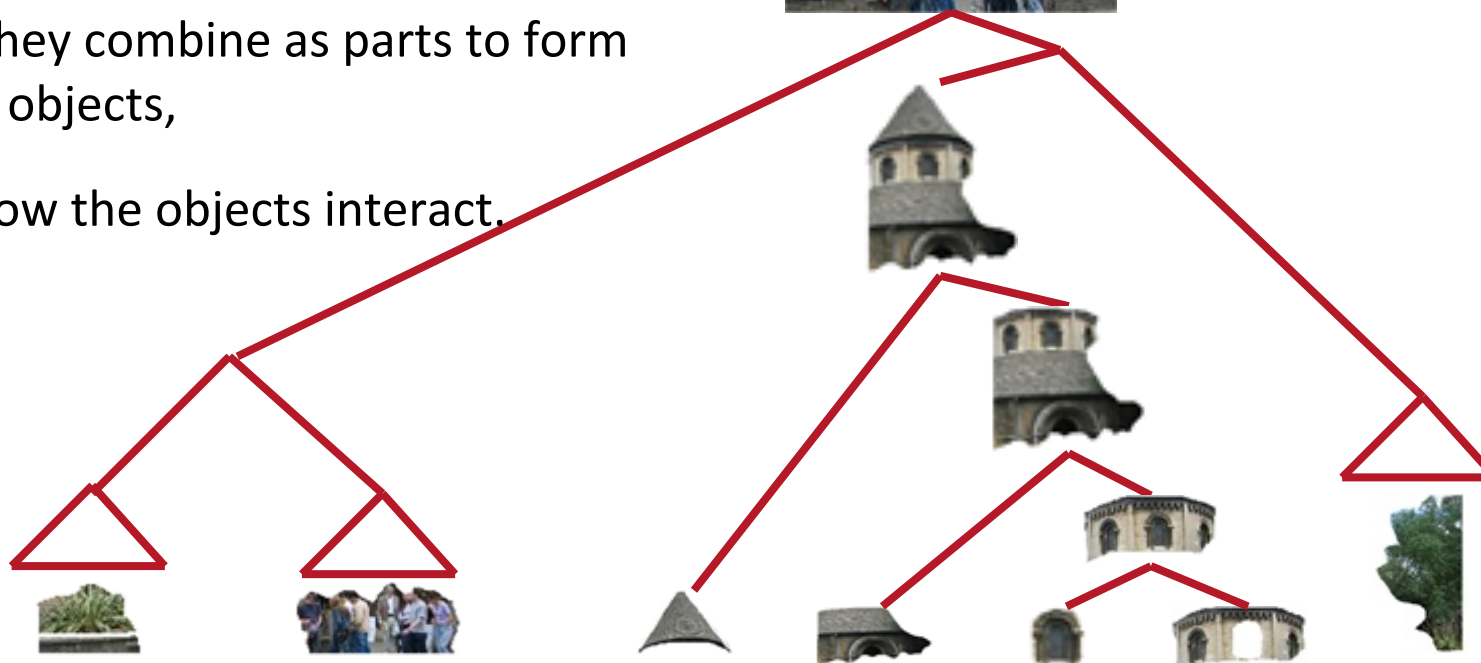


Target	Candidates with human goodness label (1–5) ordered by recursive net
the united states	the usa (5) the us (5) united states (5) north america (4) united (1) the (1) of the united states (3) america (5) nations (2) we (3)
around the world	around the globe(5) throughout the world(5) across the world(5) over the world(2) in the world(5) of the budget(2) of the world(5)
it would be	it would represent (5) there will be (2) that would be (3) it would be ideal (2) it would be appropriate (2) it is (3) it would (2)
of capital punishment	of the death penalty (5) to death (2) the death penalty (2) of (1)
in the long run	in the long term (5) in the short term (2) for the longer term (5) in the future (5) in the end (3) in the long-term (5) in time (5) of the (1)

Scene Parsing

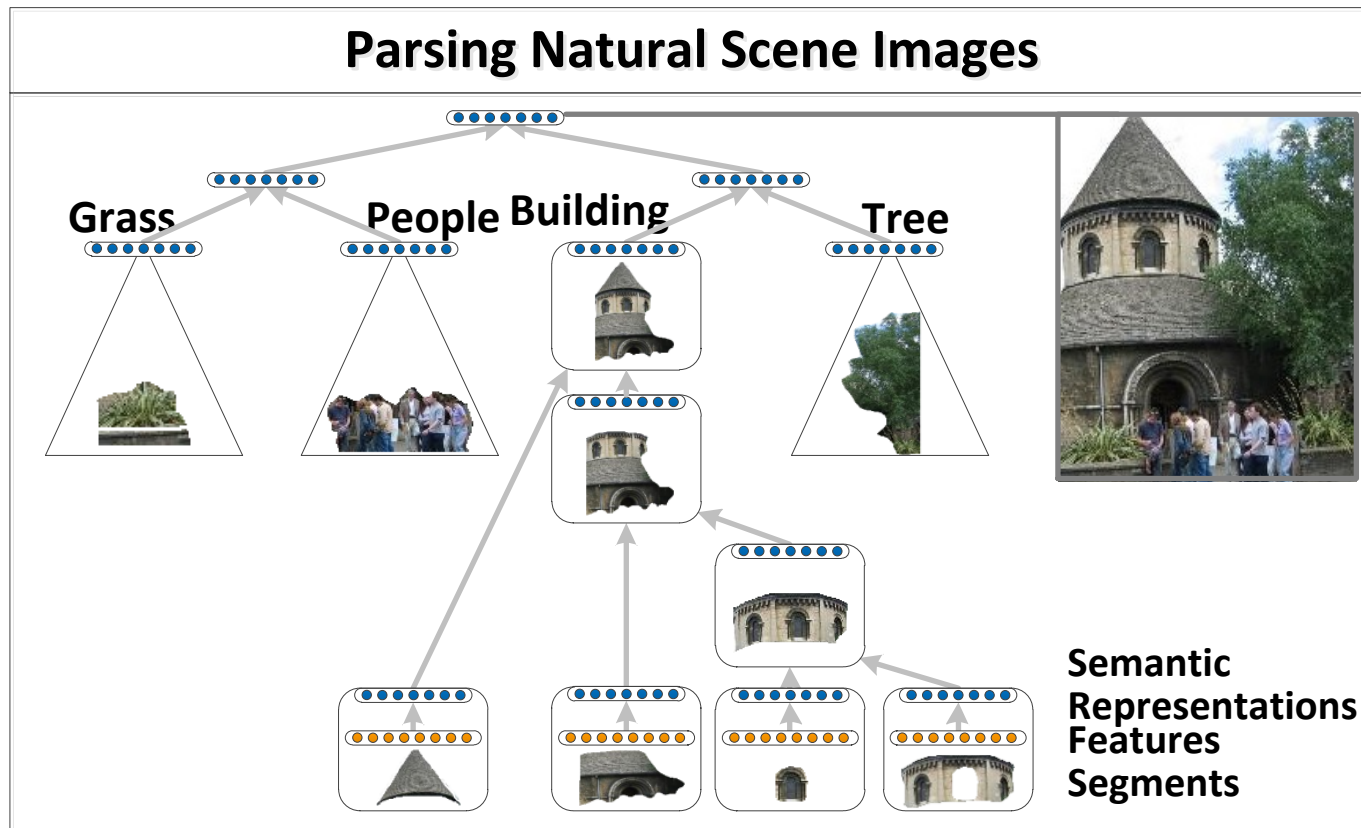
Similar principle of compositionality.

- The meaning of a scene image is also a function of smaller regions,
- how they combine as parts to form larger objects,
- and how the objects interact.



Algorithm for Parsing Images

Same Recursive Neural Network as for natural language parsing!
(Socher et al. ICML 2011)



Multi-class segmentation

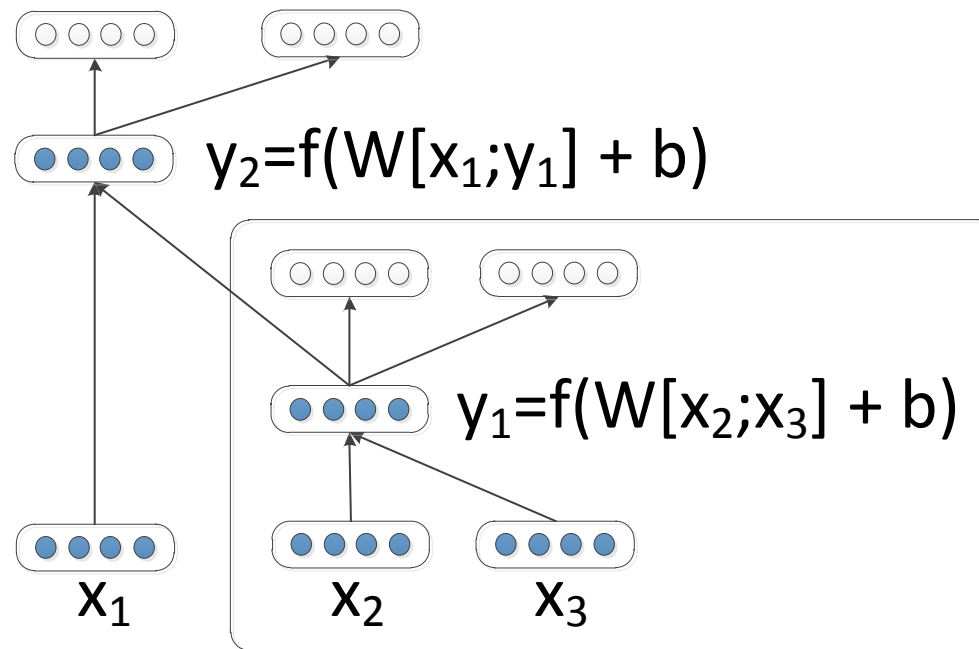


■ sky ■ tree ■ road ■ grass ■ water ■ bldg ■ mntn ■ fg obj.

Method	Accuracy
Pixel CRF (Gould et al., ICCV 2009)	74.3
Classifier on superpixel features	75.9
Region-based energy (Gould et al., ICCV 2009)	76.4
Local labelling (Tighe & Lazebnik, ECCV 2010)	76.9
Superpixel MRF (Tighe & Lazebnik, ECCV 2010)	77.5
Simultaneous MRF (Tighe & Lazebnik, ECCV 2010)	77.5
Recursive Neural Network	78.1

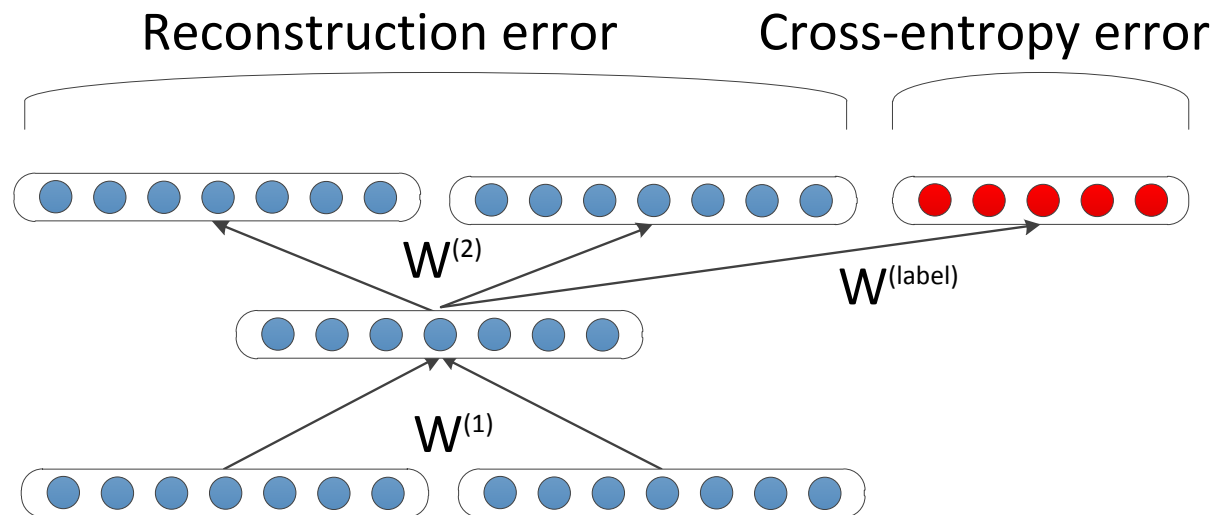
Recursive Autoencoders

- Similar to Recursive Neural Net but instead of a supervised score we compute a reconstruction error at each node. $E_{rec}([c_1; c_2]) = \frac{1}{2} ||[c_1; c_2] - [c'_1; c'_2]||^2$



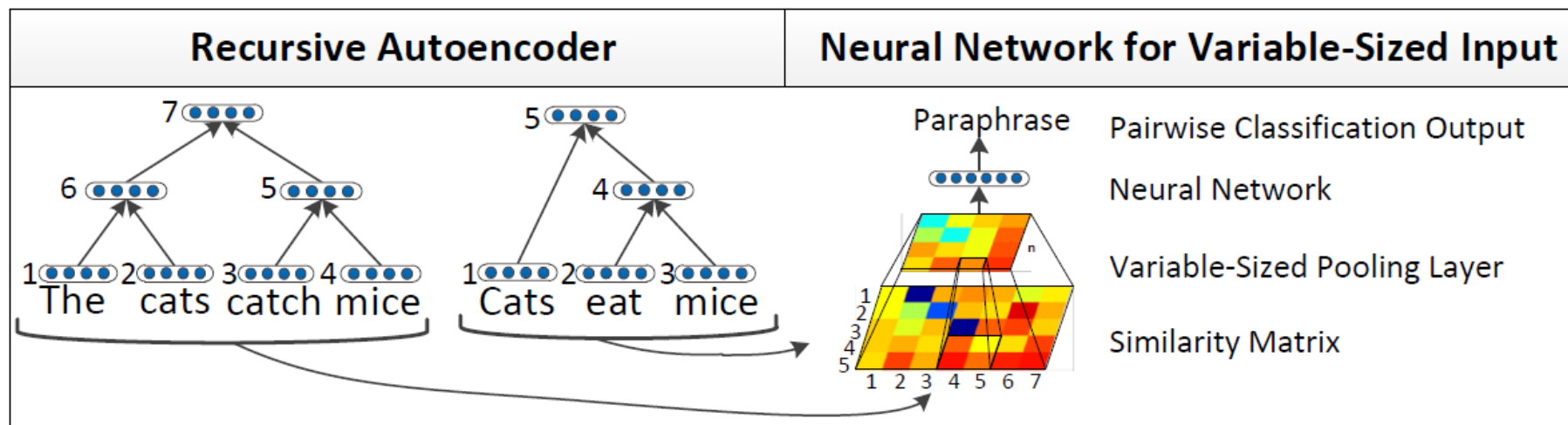
Semi-supervised Recursive Autoencoder

- To capture sentiment and solve antonym problem, add a softmax classifier
- Error is a weighted combination of reconstruction error and cross-entropy
- Socher et al. (EMNLP 2011)



Comparing the meaning of two sentences: Paraphrase Detection

- Unsupervised Unfolding RAE and a pair-wise sentence comparison of nodes in parsed trees
- Socher et al. (NIPS 2011)



Recursive Autoencoders for Full Sentence Paraphrase Detection

- Experiments on Microsoft Research Paraphrase Corpus
- (Dolan et al. 2004)

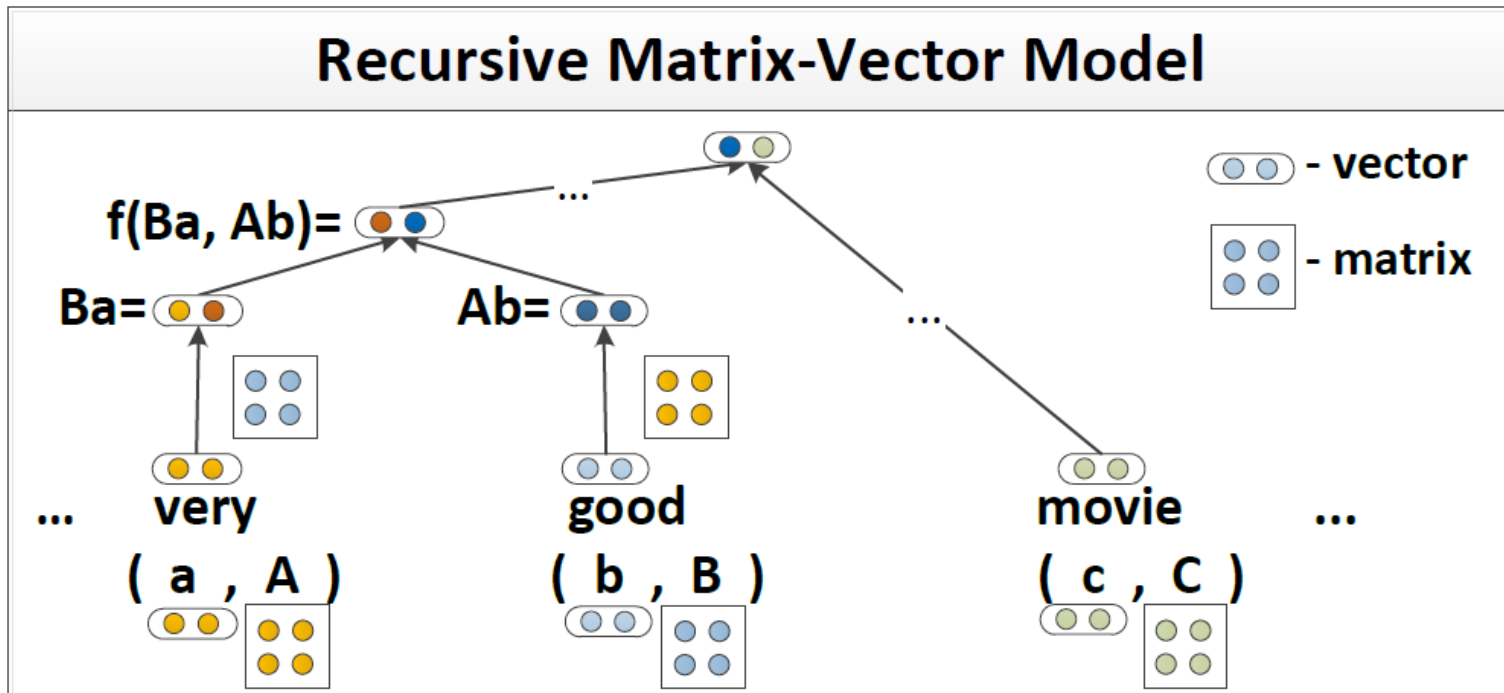
Method	Acc.	F1
Rus et al.(2008)	70.6	80.5
Mihalcea et al.(2006)	70.3	81.3
Islam et al.(2007)	72.6	81.3
Qiu et al.(2006)	72.0	81.6
Fernando et al.(2008)	74.1	82.4
Wan et al.(2006)	75.6	83.0
Das and Smith (2009)	73.9	82.3
Das and Smith (2009) + 18 Surface Features	76.1	82.7
F. Bu et al. (ACL 2012): String Re-writing Kernel	76.3	--
Unfolding Recursive Autoencoder (NIPS 2011)	76.8	83.6



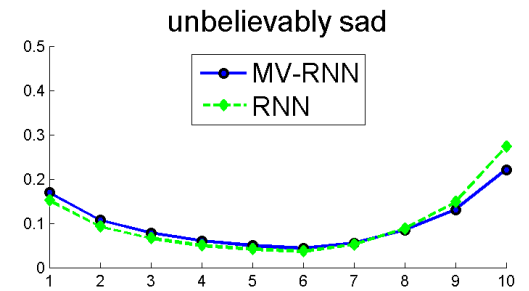
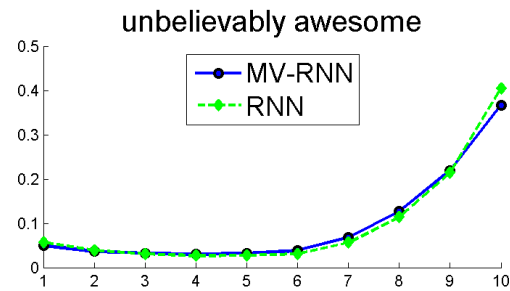
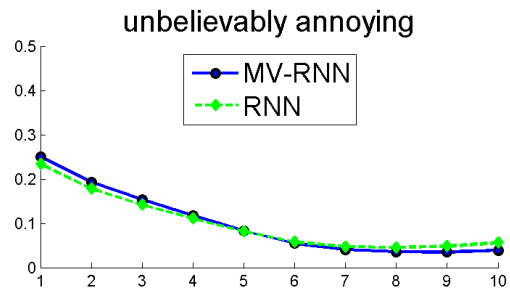
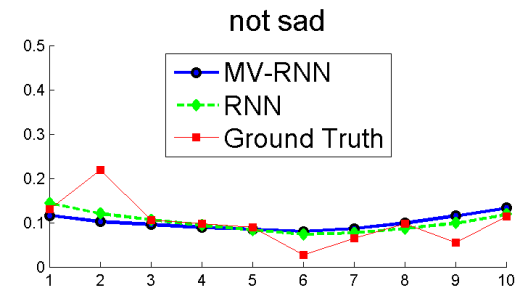
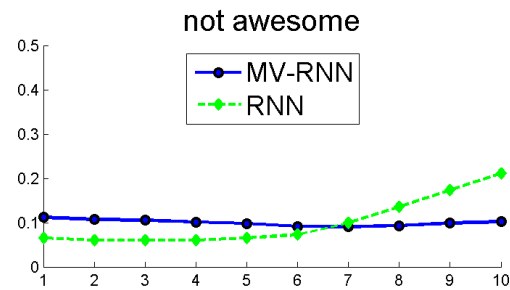
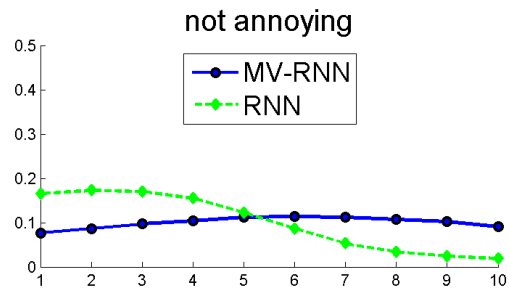
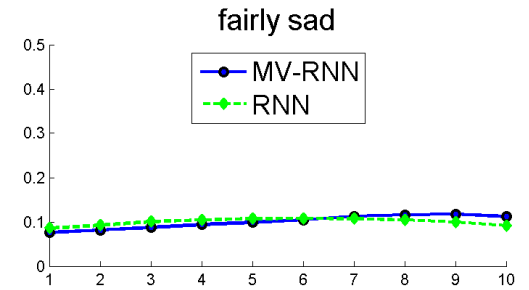
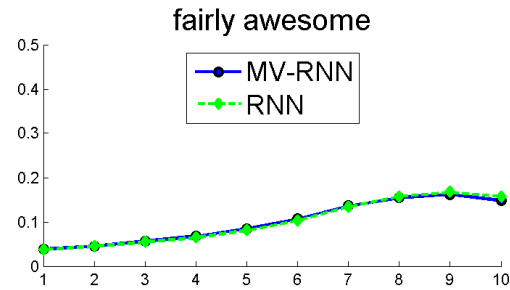
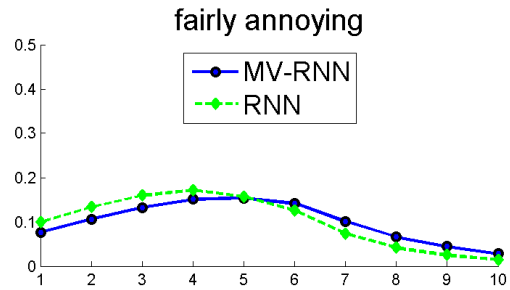
Compositionality Through Recursive Matrix-Vector Recursive Neural Networks

$$p = \tanh \left(W \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} + b \right)$$

$$p = \tanh \left(W \begin{bmatrix} C_2 c_1 \\ C_1 c_2 \end{bmatrix} + b \right)$$



Predicting Sentiment Distributions



Discussion

CONCERNS

- Many algorithms and variants (burgeoning field)
- Hyper-parameters (layer size, regularization, possibly learning rate)
 - Use multi-core machines, clusters and **random sampling for cross-validation** (Bergstra & Bengio 2012)
 - Pretty common for powerful methods, e.g. BM25

CONCERNS

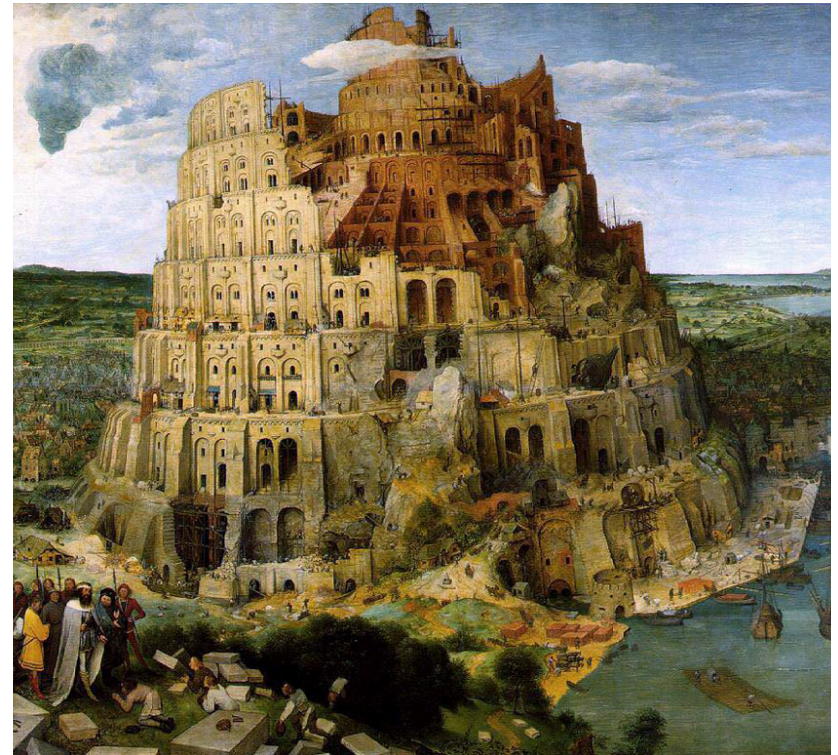
- Slower to train than linear models
 - Only by a small constant factor, and much more compact than non-parametric (e.g. n-gram models)
 - Very fast during inference/test time (feed-forward pass is just a few matrix multiplies)
- Need more training data?
 - Can *handle and benefit from* more training data (esp. unlabeled), suitable for age of Big Data (Google trains neural nets with a billion connections, [Le et al, ICML 2012])
 - Need less ***labeled*** data

Concern: non-convex optimization

- Can initialize system with convex learner
 - Convex SVM
 - Fixed feature space
- Then optimize non-convex variant (add and tune learned features), can't be worse than convex learner

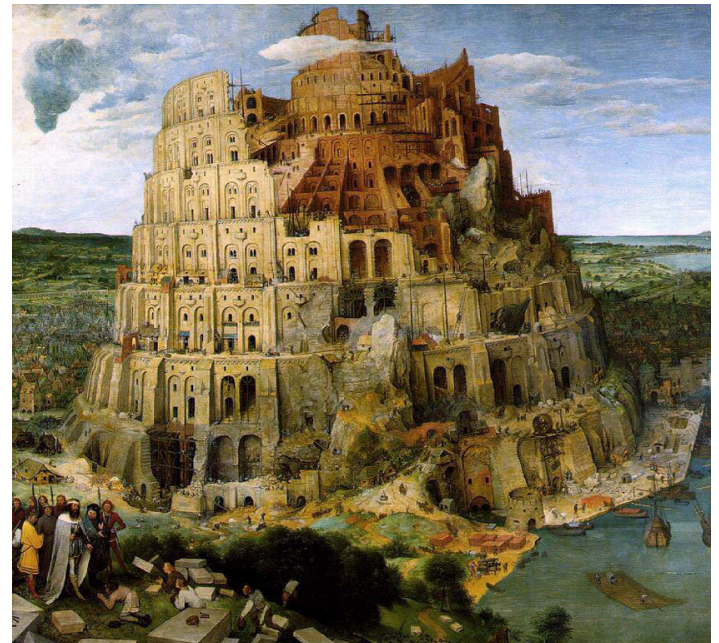
Transfer Learning

- Application of deep learning could be in areas where there are not enough labeled data but a transfer is possible
- Domain adaptation already showed that effect, thanks to **unsupervised feature learning**
- **Two transfer learning competitions won in 2011**
- Transfer to resource-poor languages would be a great application [**Gouws, PhD proposal 2012**]



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
- More abstract representations
 - Successful transfer (domains, languages)



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

Culture vs Local Minima

arXiv paper 2012, Y. Bengio

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.

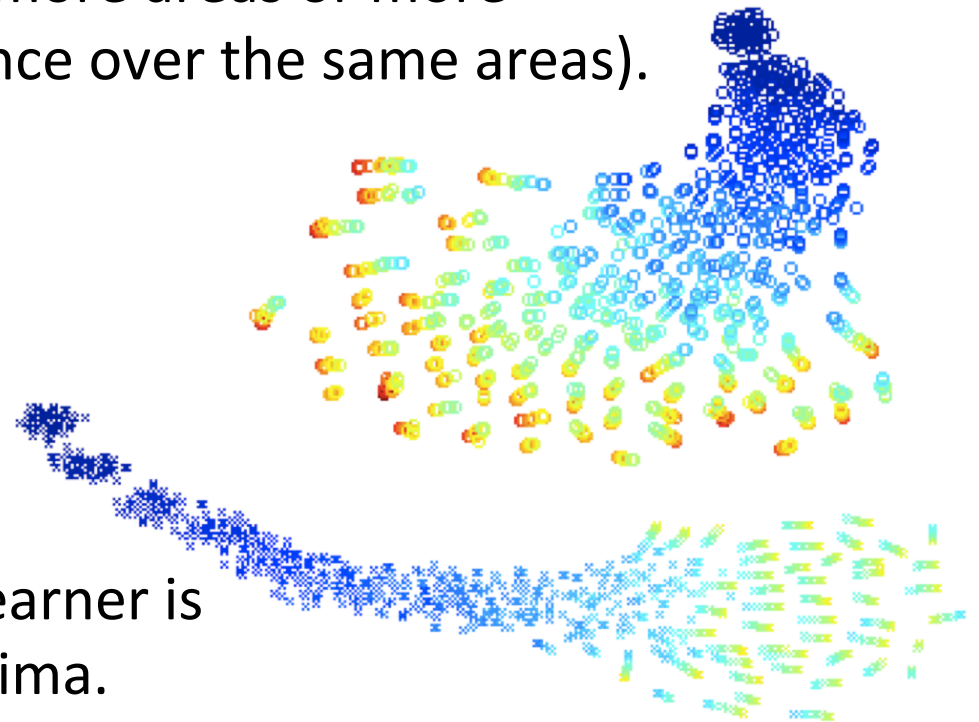
Theoretical and experimental results on deep learning suggest:

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.



Hypothesis 5

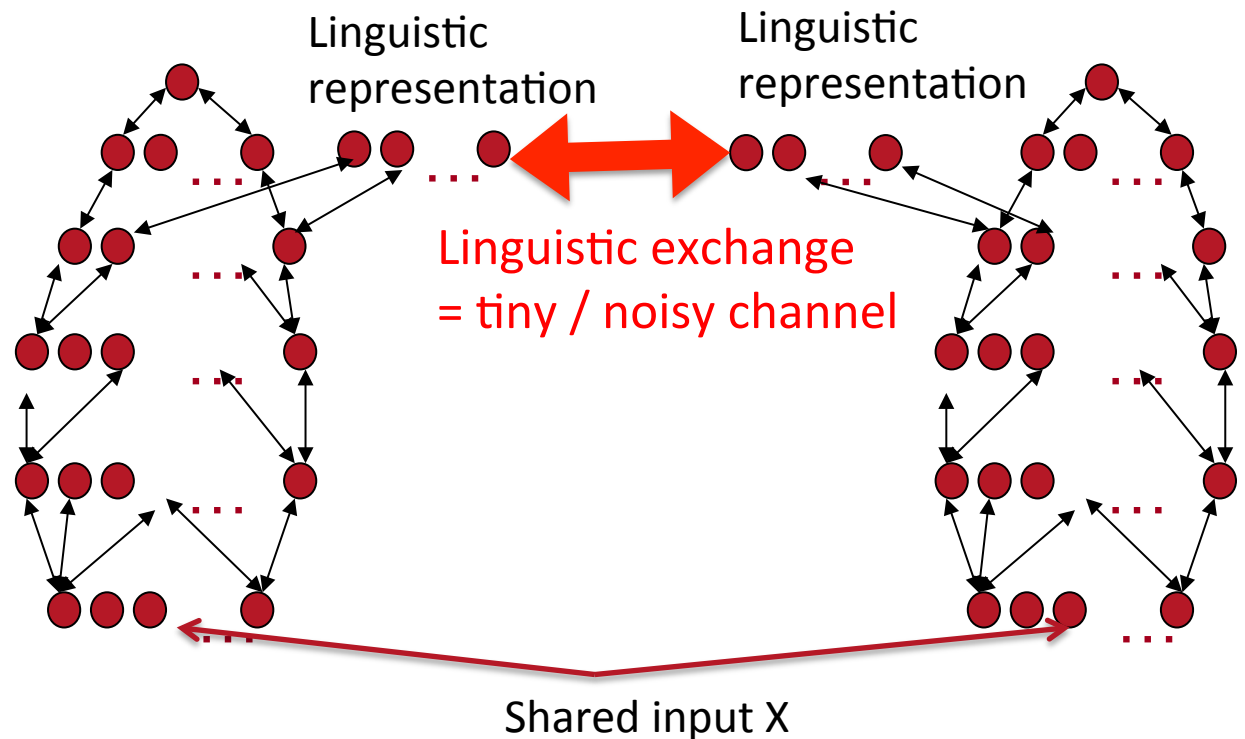
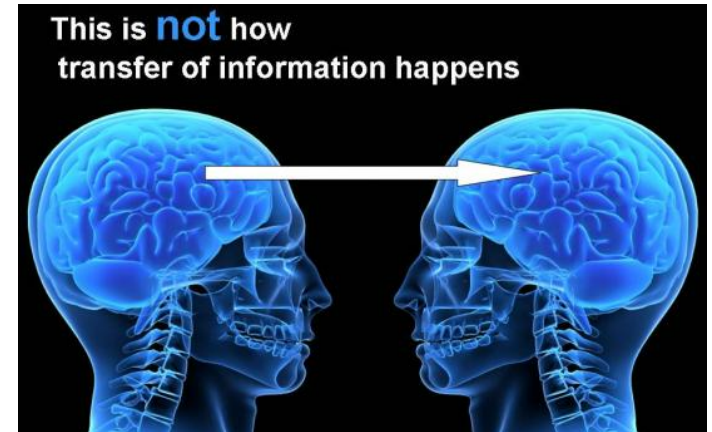
- A single human learner is unlikely to discover high-level abstractions by chance because these are represented by a deep sub-network in the brain.

Hypothesis 6

Curriculum learning

- 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.

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?

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

Hypothesis 7

- 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

- See “*Neural Net Language Models*” **Scholarpedia** entry
- Deep Learning tutorials: <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>
- **More reading: Paper references in separate pdf, on my web page**

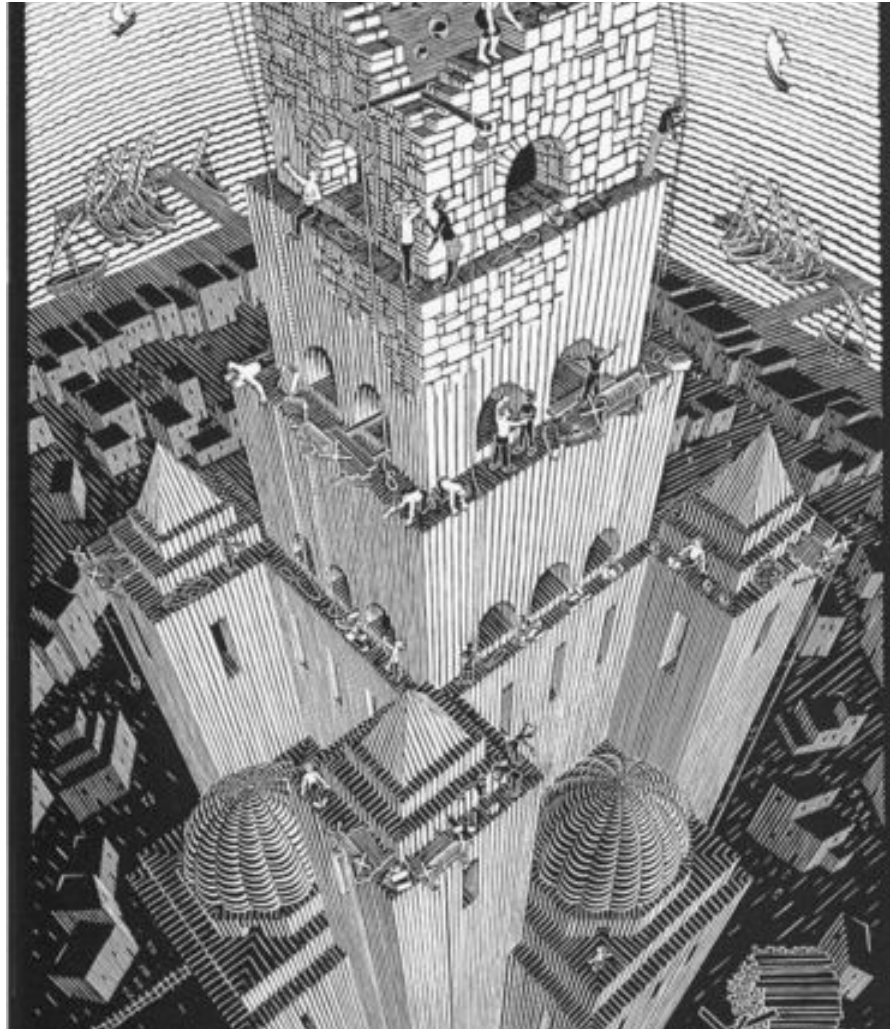
Software

- 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
- 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

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 hyperparameter optimization, exploiting multi-core or cluster at disposal of user.

The End



LISA team: **Merci! Questions?**

