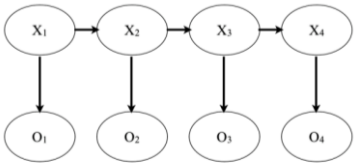
Unsupervised Deep Learning

Stats 306B

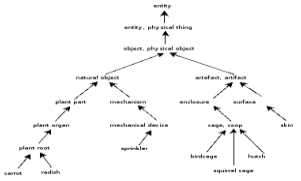
Richard Socher

Some slides from ACL 2012 Tutorial by Richard Socher, Yoshua Bengio and Chris Manning

Deep Learning

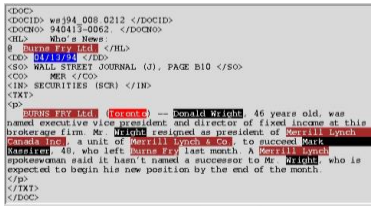


NER

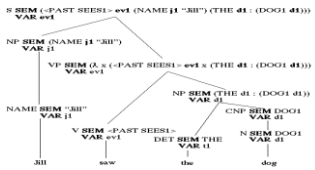


WordNet

Most current machine learning works well because of human-designed representations and input features

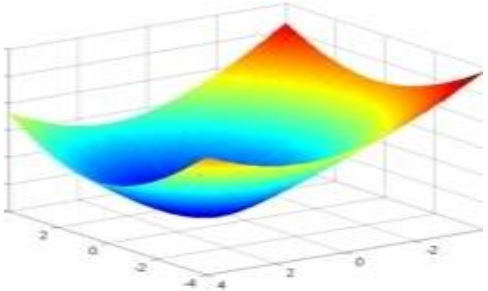


SRL



Parser

Machine learning becomes just optimizing weights to best make a final prediction



Representation learning attempts to automatically learn good features or representations

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

A Deep Architecture

Mainly, work has explored deep belief networks (DBNs), Markov Random Fields with multiple layers, and various types of multiple-layer neural networks

Output layer

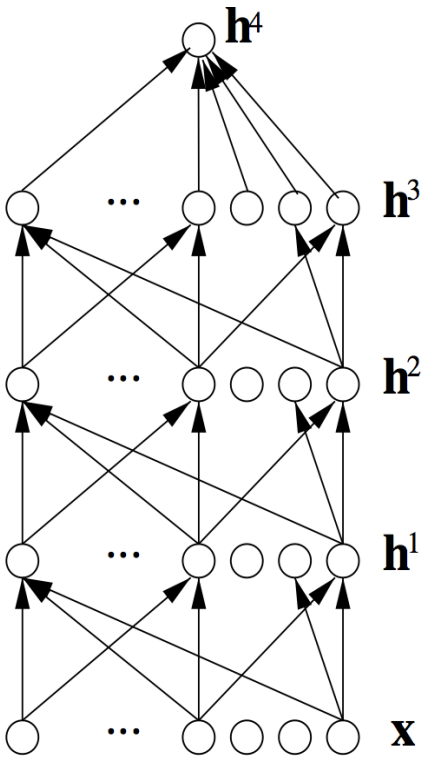
Here predicting a supervised target

Hidden layers

These learn more abstract representations as you head up

Input layer

Raw sensory inputs (roughly)



Part 1.1: The Basics

Five Reasons to Explore Deep and Representation Learning

#1 Learning representations

Handcrafting features is time-consuming

The features are often both over-specified and incomplete

The work has to be done again for each task/domain/…

We must move beyond handcrafted features and simple ML

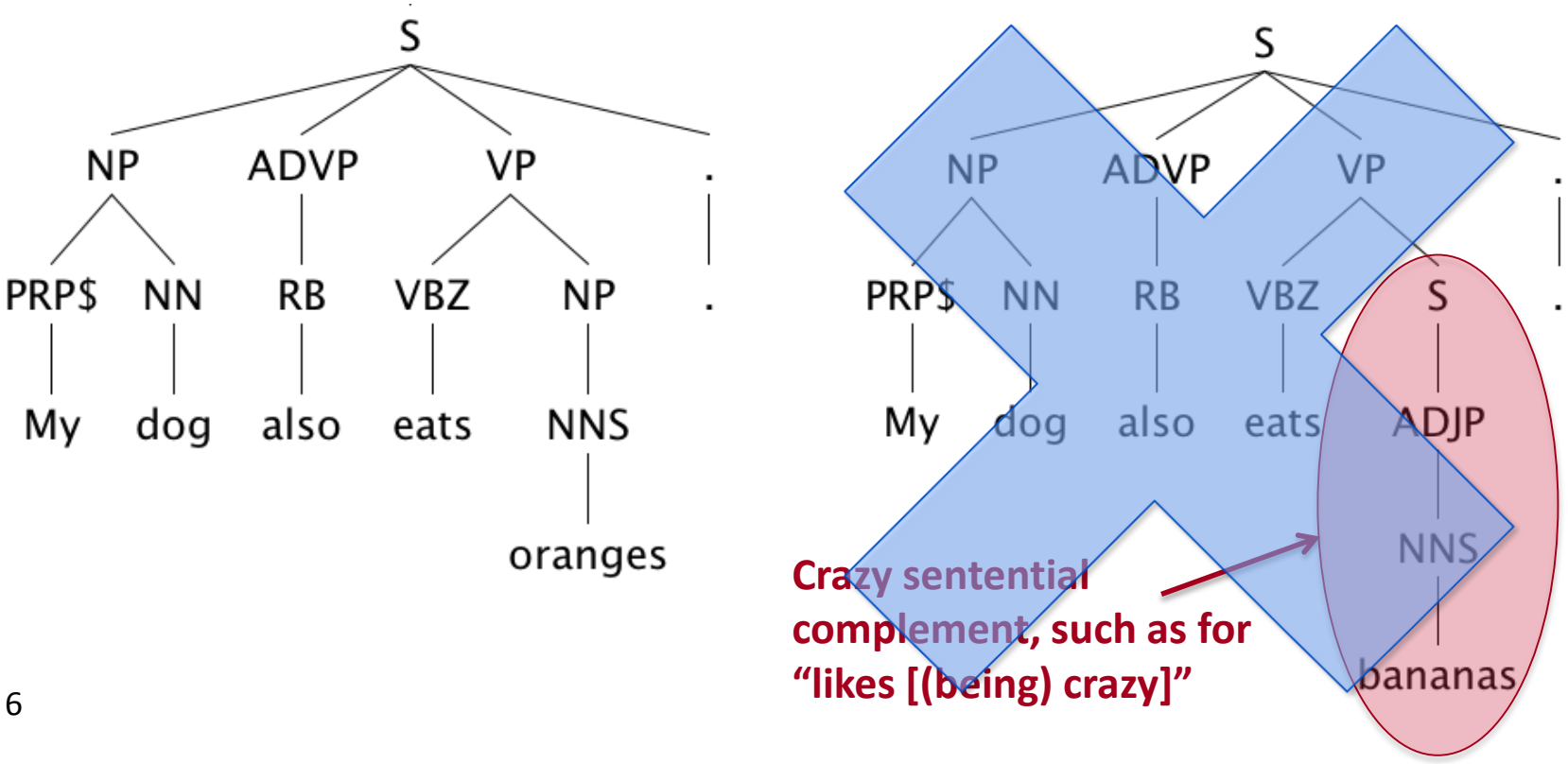
Humans develop representations for learning and reasoning

Our computers should do the same

Deep learning provides a way of doing this

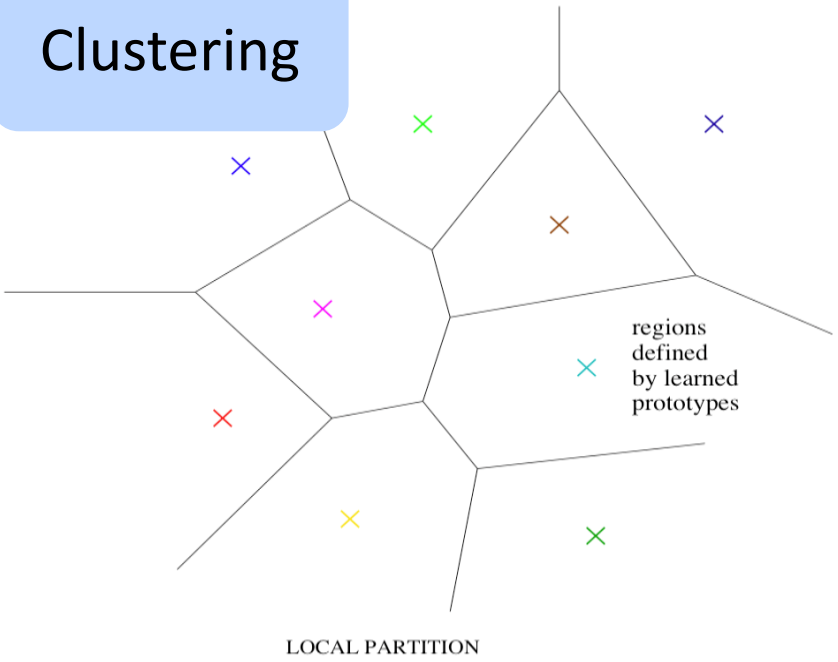
#2 The need for distributed representations

Current ML/NLP systems are incredibly fragile because of their atomic symbol representations, very sparse

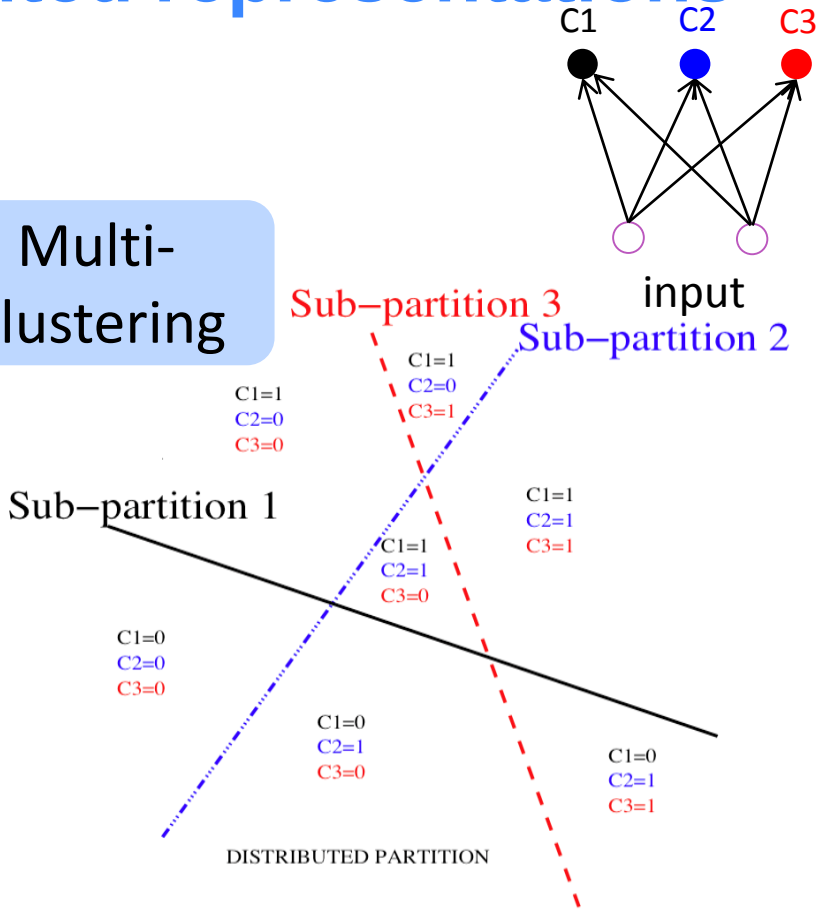


#2 The need for distributed representations

Clustering



Multi-
Clustering



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

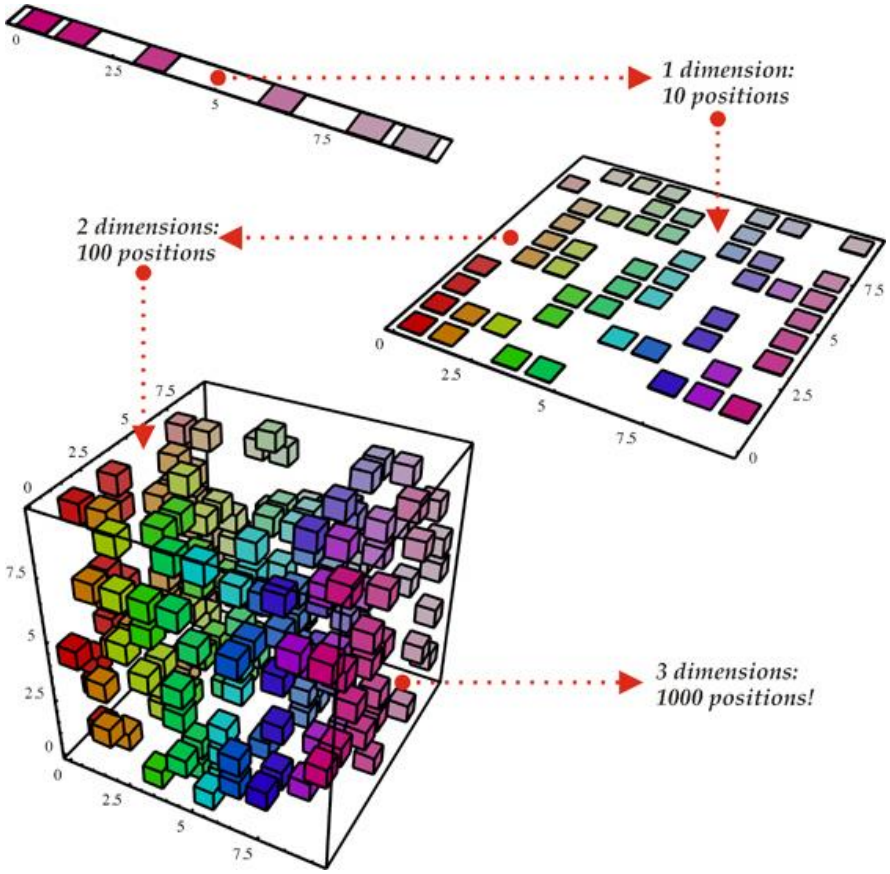
Distributed representations deal with the curse of dimensionality

Generalizing locally (e.g., nearest neighbors) requires representative examples for all relevant variations!

Classic solutions:

- Manual feature design
- Assuming a smooth target function (e.g., linear models)
- Kernel methods (linear in terms of kernel based on data points)

Neural networks parameterize and learn a “similarity” kernel



#3 Unsupervised feature and weight learning – Focus Today!

Today, most practical, good NLP& ML methods require labeled training data (i.e., supervised learning)

- But almost all data is unlabeled

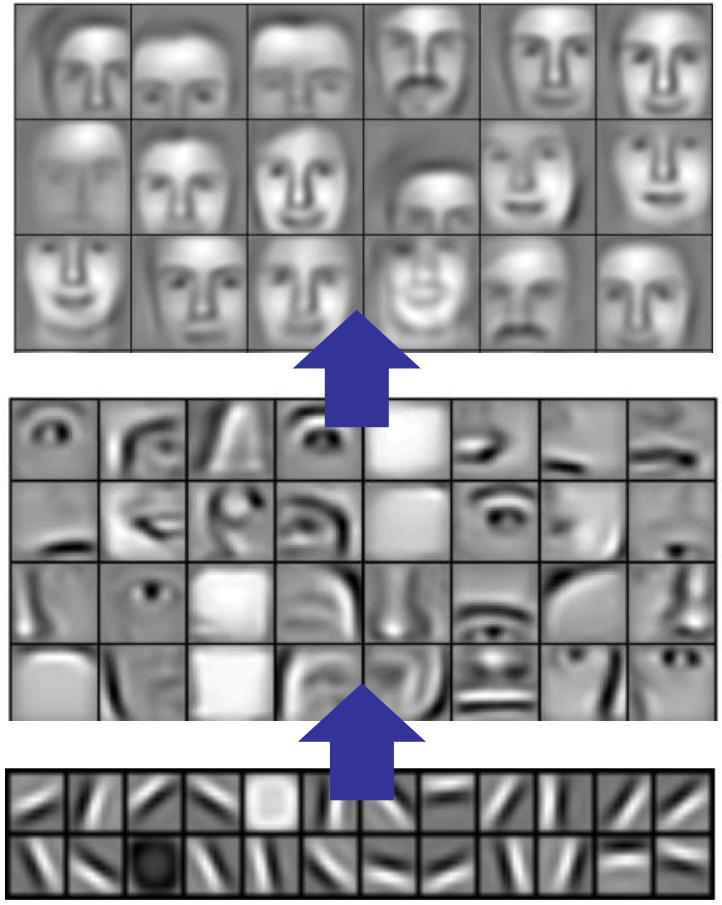
Most information must be acquired **unsupervised**

- Fortunately, a good model of observed data can really help you learn classification decisions

#4 Learning multiple levels of representation

We need good intermediate representations that can be shared across tasks

Multiple levels of latent variables allow combinatorial sharing of statistical strength

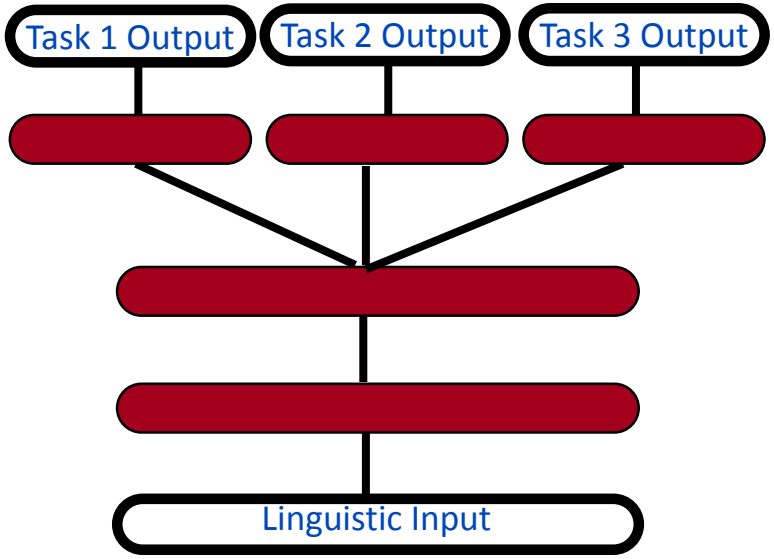


Layer 3

Layer 2

Layer 1

Biologically inspired
generic learning algorithm.



#5 Why now?

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

Before 2006 training deep architectures was **unsuccessful** ☹

What has changed?

- Faster and multicore CPUs and GPUs and more data
- New methods for unsupervised pre-training have been developed (Restricted Boltzmann Machines = RBMs, autoencoders, contrastive estimation, etc.)
- More efficient parameter estimation methods
- Better understanding of model regularization

→ Great performance in speech, vision and language tasks

Outline

1. Motivation
2. Neural Networks: Feed-forward, Autoencoders
3. Probabilistic - Directed: PCA, Sparse Coding
4. Probabilistic – Undirected: MRFs and RBMs

Part 2.2: Neural Nets Feed-forward Intro

From logistic regression to neural nets

Demystifying neural networks

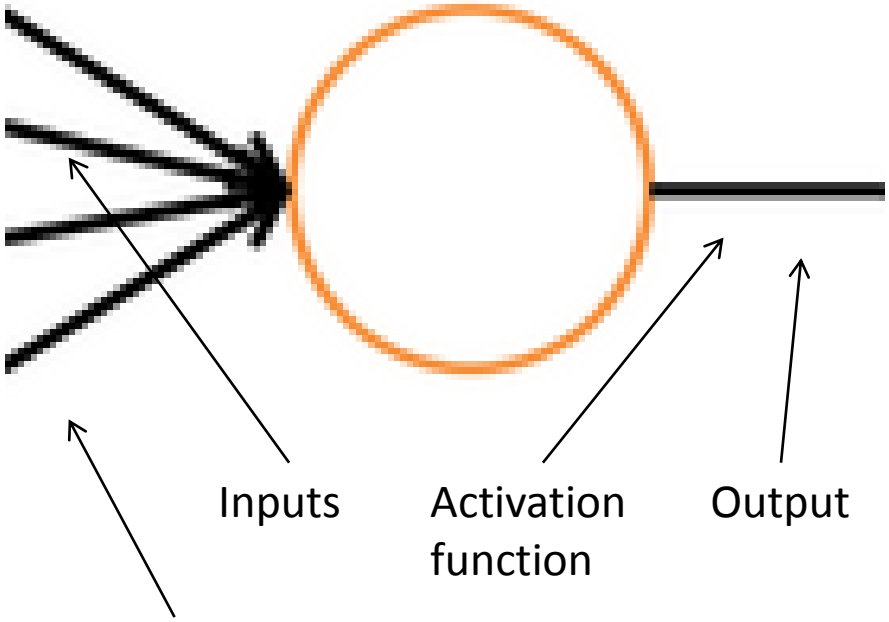
Neural networks come with their own terminological baggage

… just like SVMs

But if you understand how maxent/logistic regression models work

Then **you already understand** the operation of a basic neural network neuron!

A single neuron
A computational unit with *n* (3) inputs and 1 output and parameters *W, b*

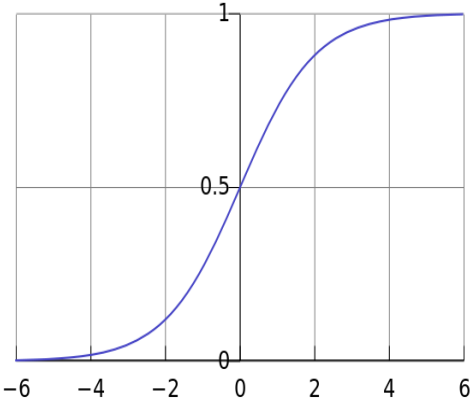


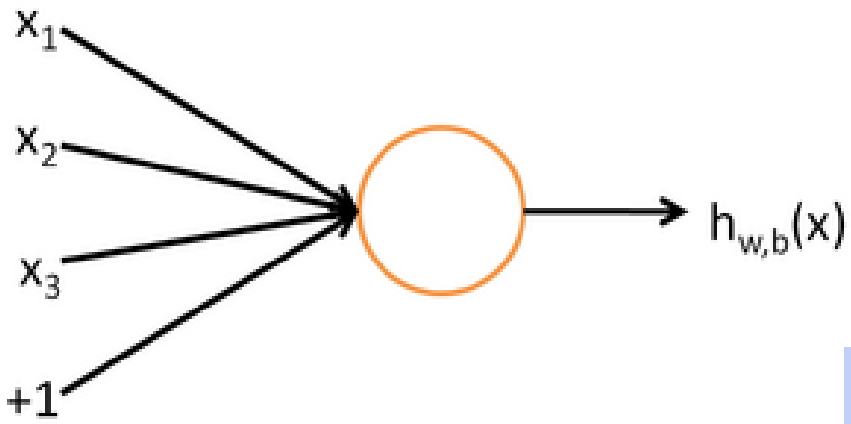
This is exactly what a neuron computes

$$h_{w,b}(x) = f(w^{\mathsf{T}}x + b)$$

b: We can have an "always on" feature, which gives a class prior, or separate it out, as a bias term

$$f(z) = \frac{1}{1+e^{-z}}$$

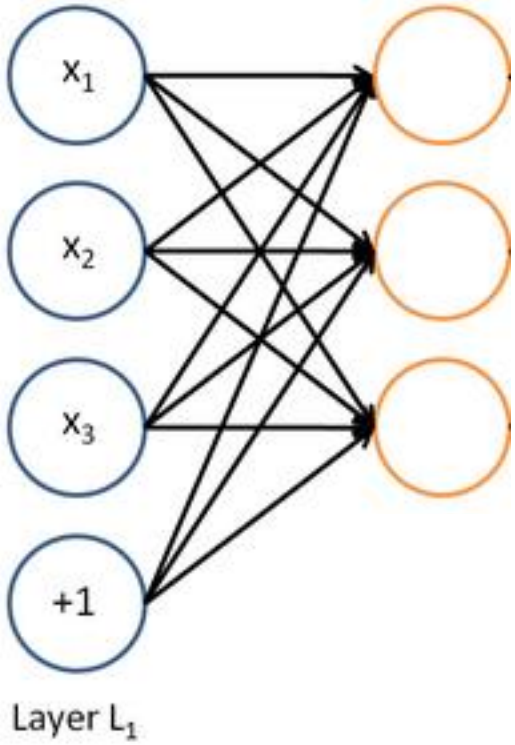




w, *b* are the parameters of this neuron i.e., this logistic regression model

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

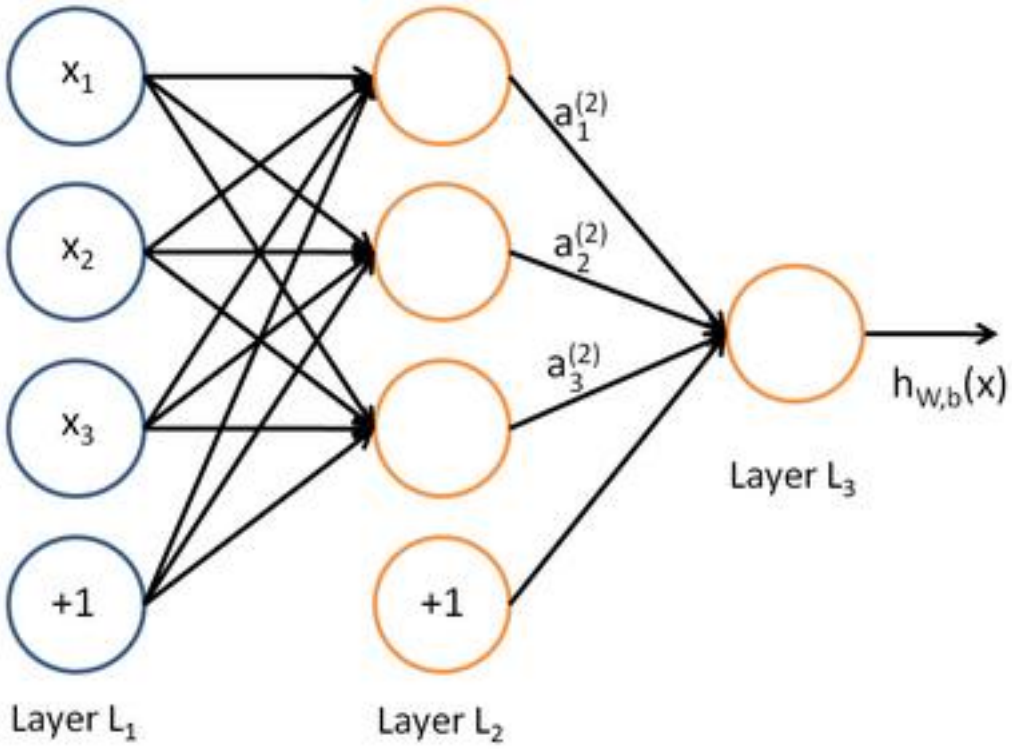
If we feed a vector of inputs through a bunch of logistic regression functions, then we get a vector of outputs



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

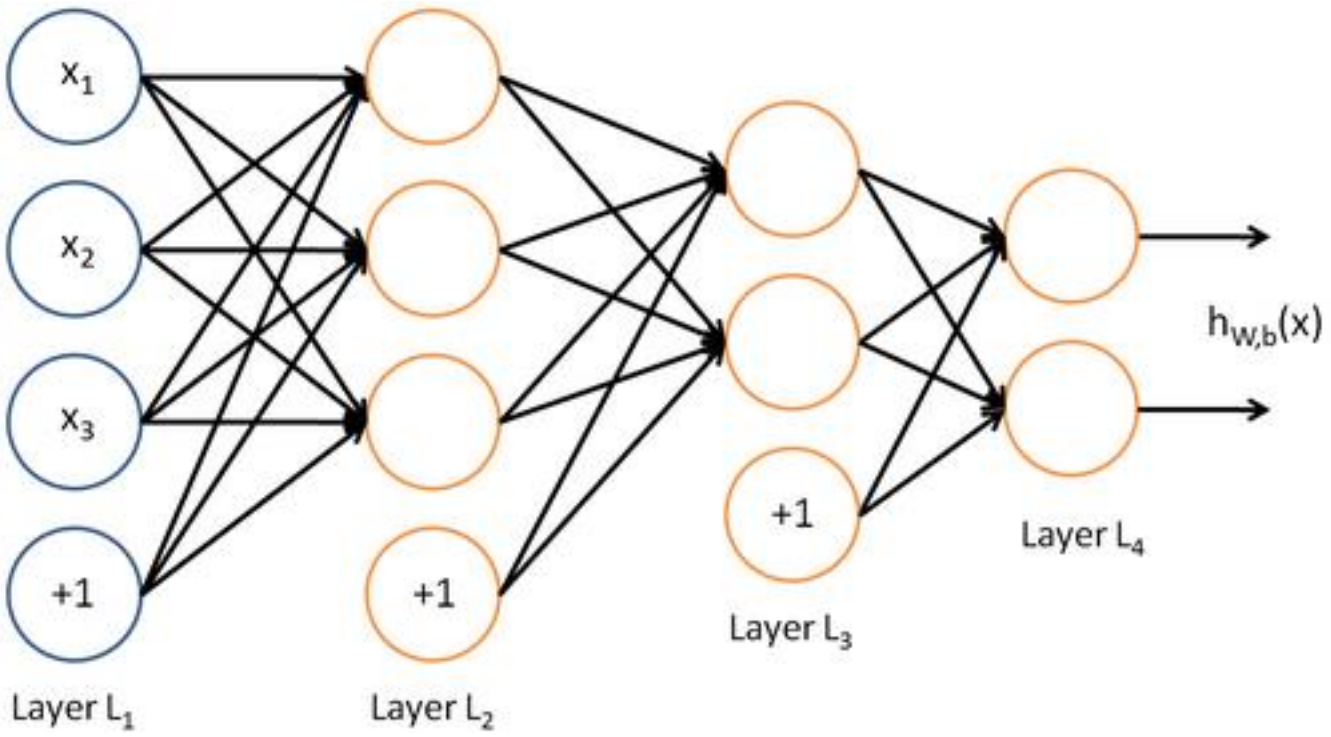
… which we can feed into another logistic regression function



and it is the training criterion that will decide what those intermediate binary target variables should be, so as to do 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….



Matrix notation for a layer

We have

$$a_1 = f(W_{11}x_1 + W_{12}x_2 + W_{13}x_3 + b_1)$$

$$a_2 = f(W_{21}x_1 + W_{22}x_2 + W_{23}x_3 + b_2)$$

etc.

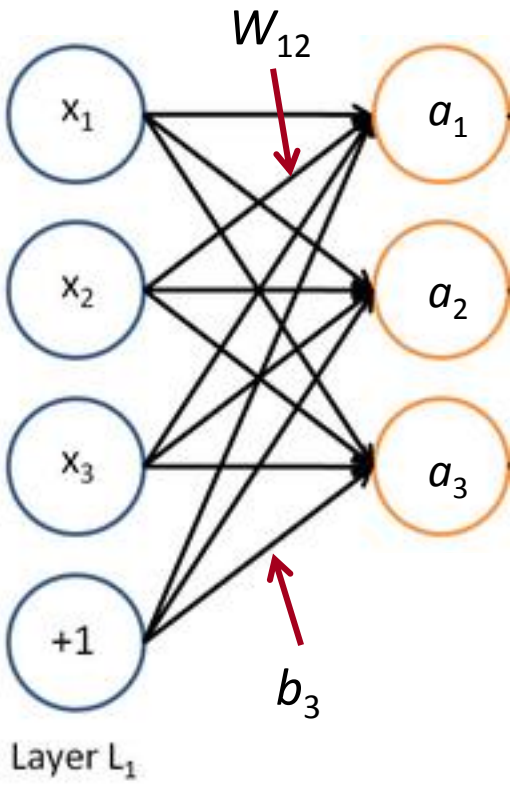
In matrix notation

$$z = Wx + b$$

$$a = f(z)$$

where f is applied element-wise:

$$f([z_1, z_2, z_3]) = [f(z_1), f(z_2), f(z_3)]$$

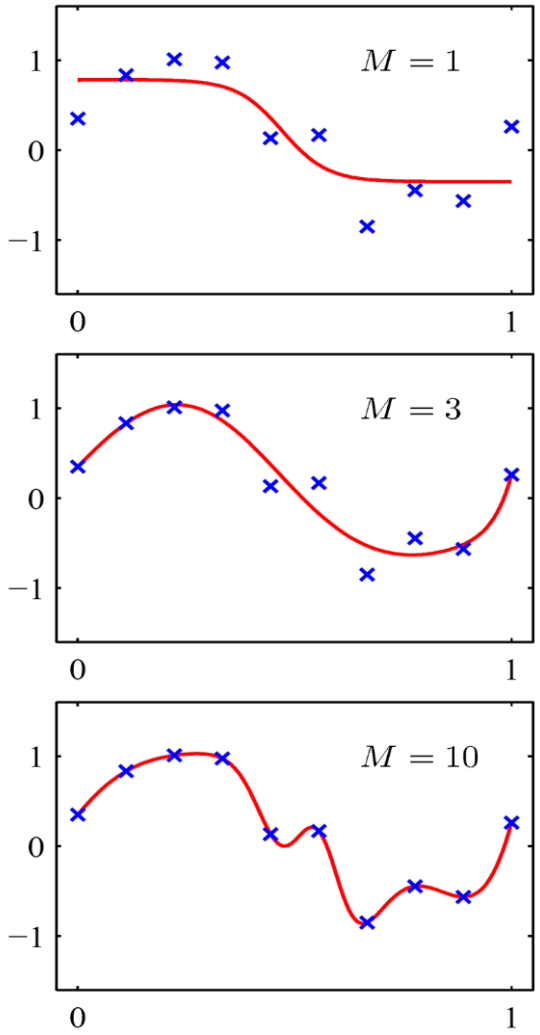


How do we train the weights *W*?

- For a supervised single layer neural net, we can train the model just like a maxent model – we calculate and use gradients
  - Stochastic gradient descent (SGD)
  - Conjugate gradient or L-BFGS
  - Adagrad!

Non-linearities: Why they're needed

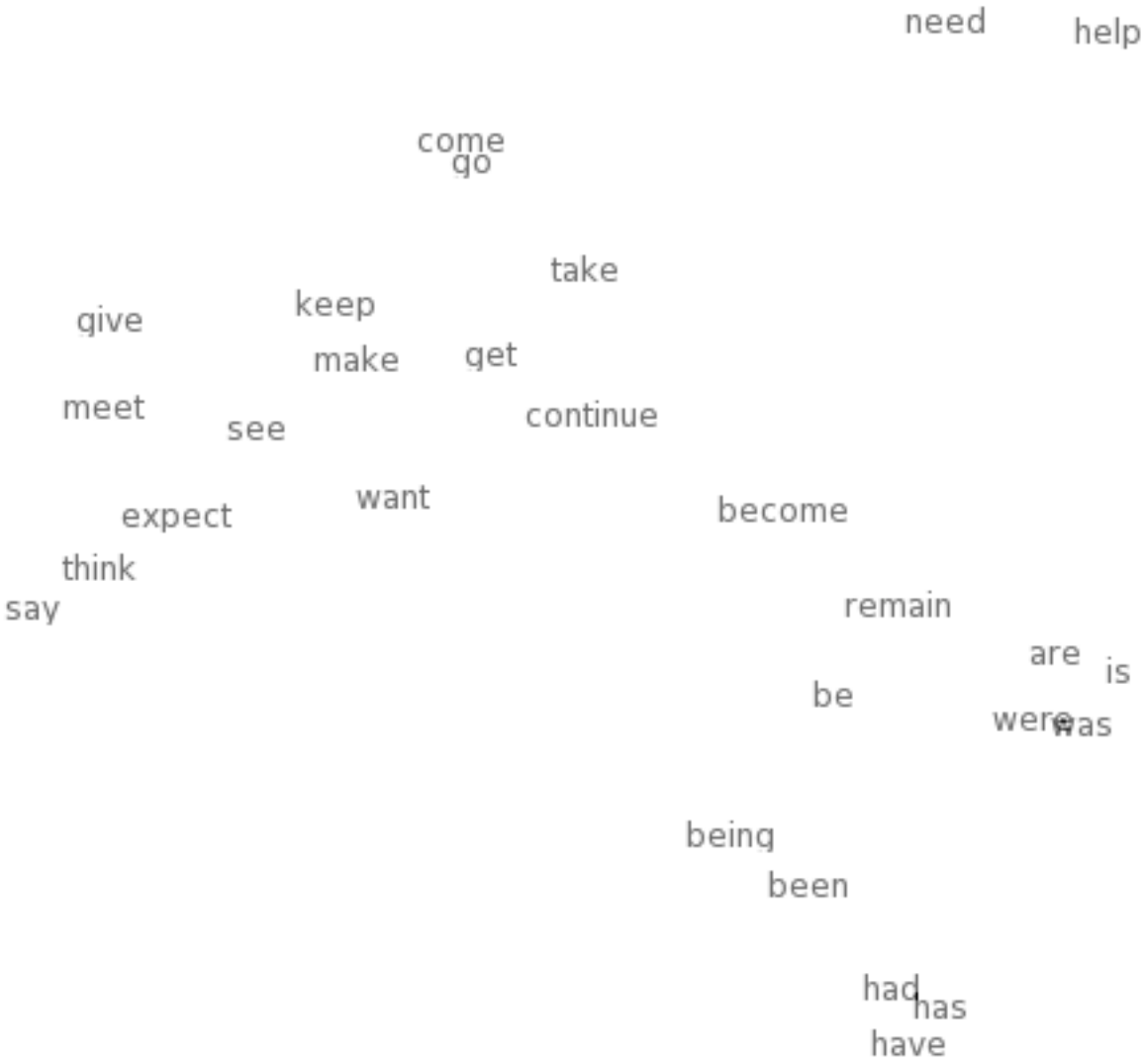
- For logistic regression: map to probabilities
- Here: function approximation,
  e.g., regression or classification
  - Without non-linearities, deep neural networks can't do anything more than a linear transform
    - Extra layers could just be compiled down into a single linear transform
  - Probabilistic interpretation unnecessary except in the Boltzmann machine/graphical models
    - People often use other non-linearities, such as **tanh**, as we'll discuss in part 3



Part 2.2: Neural Nets Feed-forward Example Model

Learning Word Representations with a Simple "Unsupervised" Model

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

A neural network for learning word vectors (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)

Summary: Feed-forward Computation

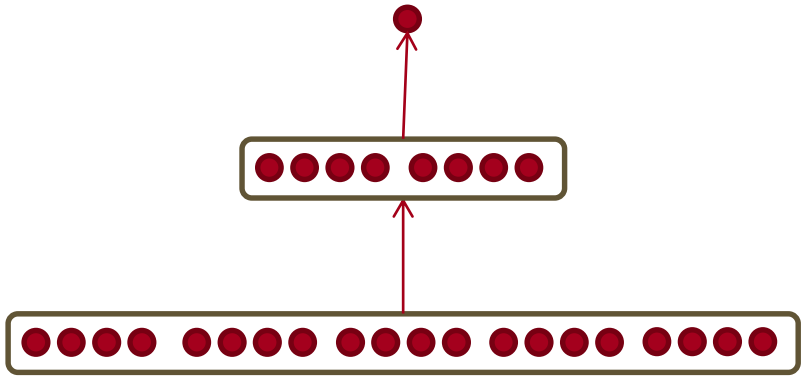
Computing a window's score with a 3-layer Neural Net: *s = score*(cat chills on a mat)

$$s = U^T f(Wx + b) \qquad x \in \mathbb{R}^{20\times 1}, W \in \mathbb{R}^{8\times 20}, U \in \mathbb{R}^{8\times 1}$$

$$\begin{aligned} s &= U^T a \\ a &= f(z) \\ z &= Wx + b \end{aligned}$$

$$x = [x_{cat} \quad x_{chills} \quad x_{on} \quad x_a \quad x_{mat}]$$

$$L \in \mathbb{R}^{n\times|V|}$$

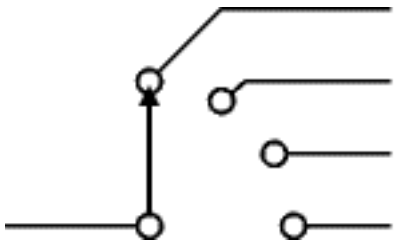


cat chills on a mat

Summary: Feed-forward Computation

- s = score(cat chills on a mat)
- s_c = score(cat chills Jeju a mat)
- Idea for training objective: make score of true window larger and **c**orrupt window's score lower (until they're good enough): minimize

$$J = \max(0, 1 - s + s_c)$$



- This is continuous, can perform SGD

Training with Backpropagation

$$J = \max(0, 1 - s + s_c)$$

$$s = U^T f(Wx + b)$$
$$s_c = U^T f(Wx_c + b)$$

Assuming cost *J* is > 0, it is simple to see that we can compute the derivatives of *s* and s_c wrt all the involved variables: *U, W, b, x*

$$\frac{\partial s}{\partial U} = \frac{\partial}{\partial U} U^T a \qquad \frac{\partial s}{\partial U} = a$$

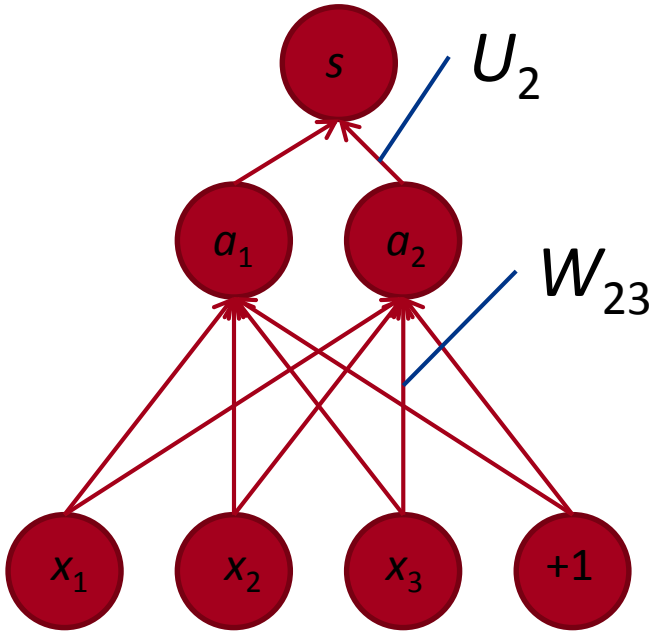
Training with Backpropagation

- Let's consider the derivative of a single weight W_{ij}

$$\frac{\partial s}{\partial W} = \frac{\partial}{\partial W} U^T a = \frac{\partial}{\partial W} U^T f(z) = \frac{\partial}{\partial W} U^T f(Wx + b)$$

- This only appears inside a_i

- For example: W_{23} is only used to compute a_2



Training with Backpropagation

$$\frac{\partial s}{\partial W} = \frac{\partial}{\partial W} U^T a = \frac{\partial}{\partial W} U^T f(z) = \frac{\partial}{\partial W} U^T f(Wx + b)$$

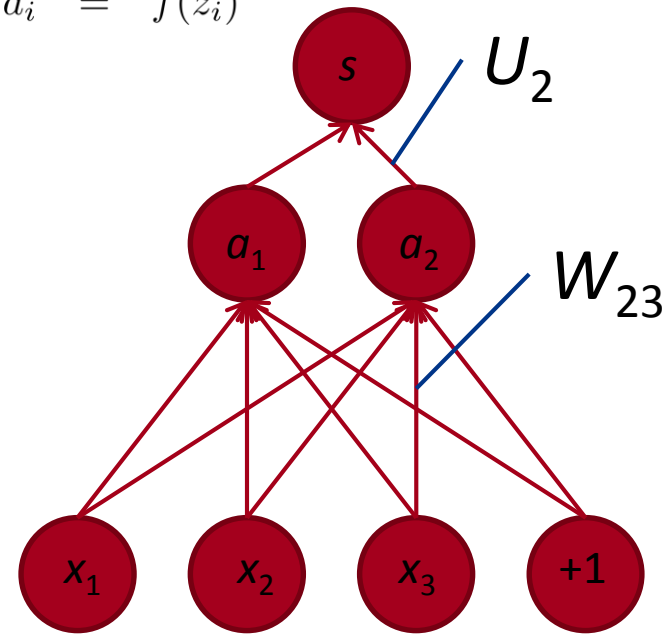
Derivative of weight W_{ij}:

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

$$\frac{\partial}{\partial W_{ij}} U^T a \quad \rightarrow \quad \frac{\partial}{\partial W_{ij}} U_i a_i$$

$$\begin{aligned} U_i \frac{\partial}{\partial W_{ij}} a_i &= U_i \frac{\partial a_i}{\partial z_i} \frac{\partial z_i}{\partial W_{ij}} \\ &= U_i \frac{\partial f(z_i)}{\partial z_i} \frac{\partial z_i}{\partial W_{ij}} \\ &= U_i f'(z_i) \frac{\partial z_i}{\partial W_{ij}} \\ &= U_i f'(z_i) \frac{\partial W_i . x + b_i}{\partial W_{ij}} \end{aligned}$$

$$\begin{aligned} z_i &= W_i . x + b_i = \sum_{j=1}^{3} W_{ij} x_j + b_i \\ a_i &= f(z_i) \end{aligned}$$



Training with Backpropagation

Derivative of single weight W_{ij} :

$$U_i \frac{\partial}{\partial W_{ij}} a_i$$

$$= U_i f'(z_i) \frac{\partial W_i.x + b_i}{\partial W_{ij}}$$

$$= U_i f'(z_i) \frac{\partial}{\partial W_{ij}} \sum_k W_{ik} x_k$$

$$= \underbrace{U_i f'(z_i)}_{\delta_i} x_j$$

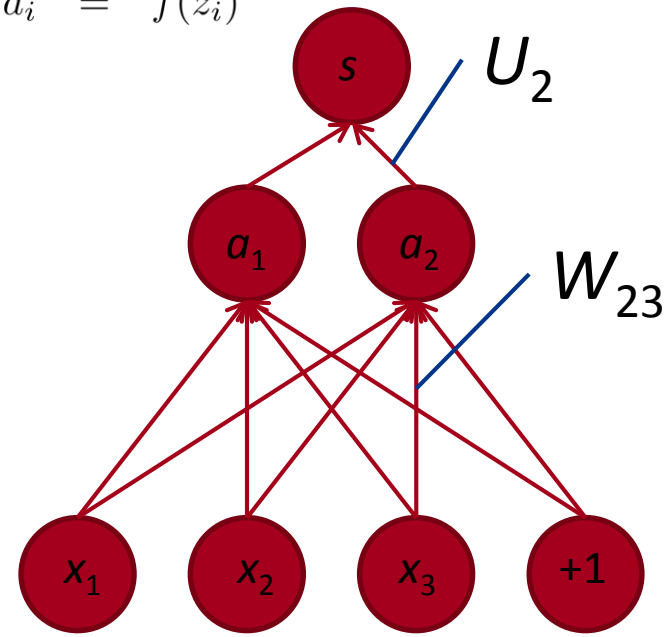
$$= \delta_i \; x_j$$

Local error signal

Local input signal

$$z_i = W_i.x + b_i = \sum_{j=1}^{3} W_{ij} x_j + b_i$$

$$a_i = f(z_i)$$



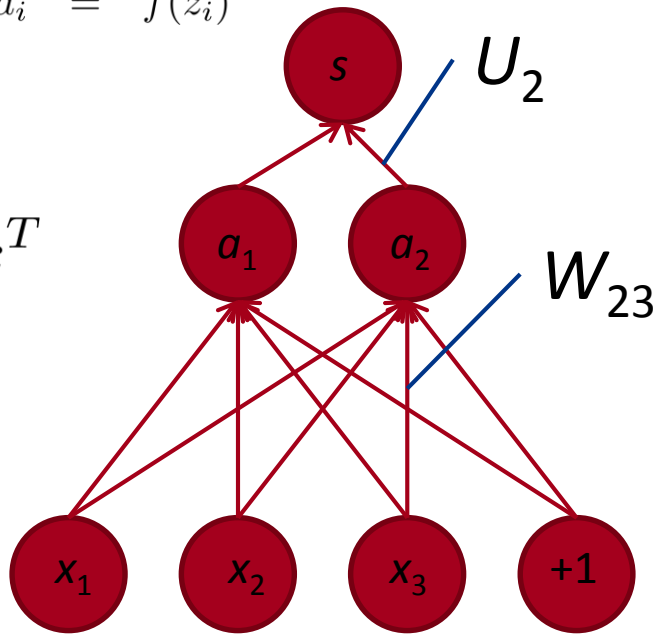
Training with Backpropagation

- From single weight W_{ij} to full W:

$$\begin{aligned}\frac{\partial J}{\partial W_{ij}} &= \underbrace{U_i f'(z_i)}_{\delta_i} x_j \\ &= \delta_i \quad x_j\end{aligned}$$

- We want all combinations of $i = 1, 2$ and $j = 1, 2, 3$
- Solution: Outer product: $\frac{\partial J}{\partial W} = \delta x^T$ where $\delta \in \mathbb{R}^{2\times 1}$ is the "responsibility" coming from each activation a

$$\begin{aligned} z_i &= W_i.x + b_i = \sum_{j=1}^{3} W_{ij}x_j + b_i \\ a_i &= f(z_i) \end{aligned}$$

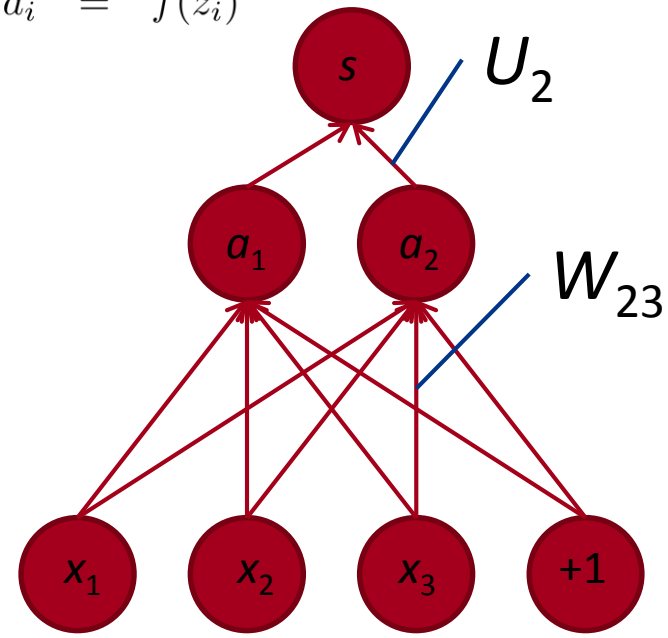


Training with Backpropagation

- For biases *b*, we get:

$$
\begin{aligned}
& U_i \frac{\partial}{\partial b_i} a_i \\
= \quad & U_i f'(z_i) \frac{\partial W_i.x + b_i}{\partial b_i} \\
= \quad & \delta_i
\end{aligned}
$$

$$
\begin{aligned}
z_i &= W_i.x + b_i = \sum_{j=1}^{3} W_{ij} x_j + b_i \\
a_i &= f(z_i)
\end{aligned}
$$



Training with Backpropagation

That's almost backpropagation

It's simply taking derivatives and using the chain rule!

Remaining trick: we can re-use derivatives computed for higher layers in computing derivatives for lower layers

Example: last derivatives of model, the word vectors in *x*

Training with Backpropagation

- Take derivative of score with respect to single word vector (for simplicity a 1d vector, but same if it was longer)
- Now, we cannot just take into consideration one a_i because each x_j is connected to all the neurons above and hence x_j influences the overall score through all of these, hence:

$$
\begin{aligned}
\frac{\partial s}{\partial x_j} &= \sum_{i=1}^{2} \frac{\partial s}{\partial a_i} \frac{\partial a_i}{\partial x_j} \\
&= \sum_{i=1}^{2} \frac{\partial U^T a}{\partial a_i} \frac{\partial a_i}{\partial x_j} \\
&= \sum_{i=1}^{2} U_i \frac{\partial f(W_i.x + b)}{\partial x_j} \\
&= \sum_{i=1}^{2} \underbrace{U_i f'(W_i.x + b)}_{} \frac{\partial W_i.x}{\partial x_j} \\
&= \sum_{i=1}^{2} \delta_i W_{ij} \\
&= \underbrace{\delta^T}_{} W_{\cdot j}
\end{aligned}
$$

Re-used part of previous derivative

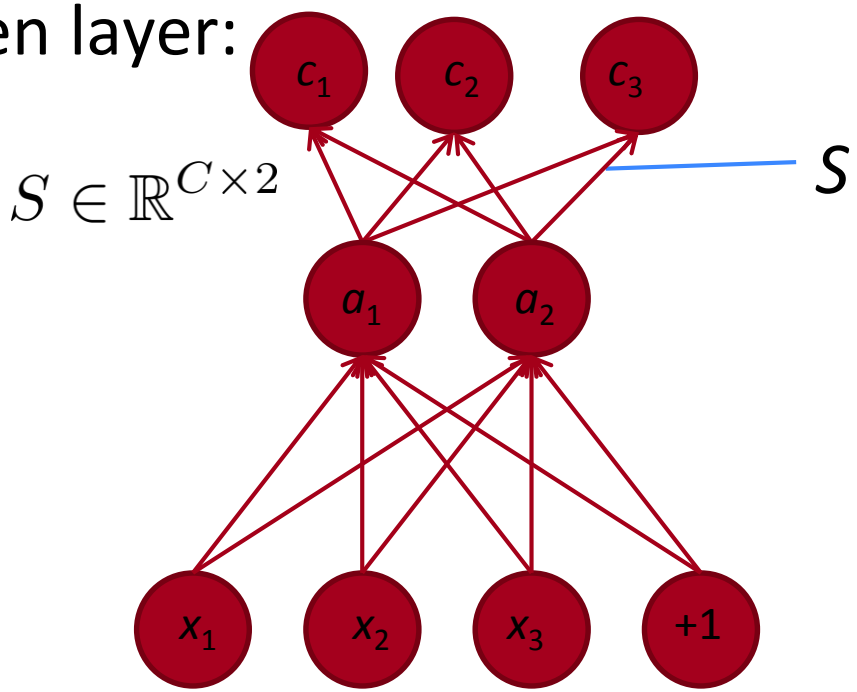
Training with Backpropagation: softmax

What is the major benefit of learned word vectors?

Ability to also propagate labeled information into them, via softmax/maxent and hidden layer:

$$P(c \mid d, l) = \frac{e^{l^\top f(c,d)}}{\sum_{c'} e^{l^\top f(c',d)}}$$

$$p(c|x) = \frac{\exp(S_c.a)}{\sum_{c'} \exp(S_{c'}.a)}$$



Part 2.2: Autoencoders

Autoencoder and their variants

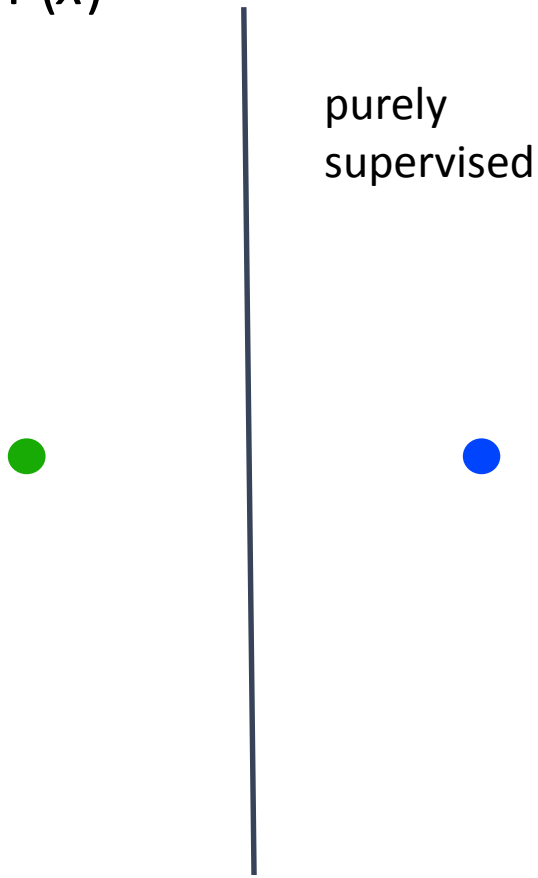
Sharing Statistical Strength

- Besides very fast prediction, the main advantage of deep learning is statistical

- Potential to learn from less labeled examples because of sharing of statistical strength:
  - Unsupervised pre-training & Multi-task learning
  - Semi-supervised learning →

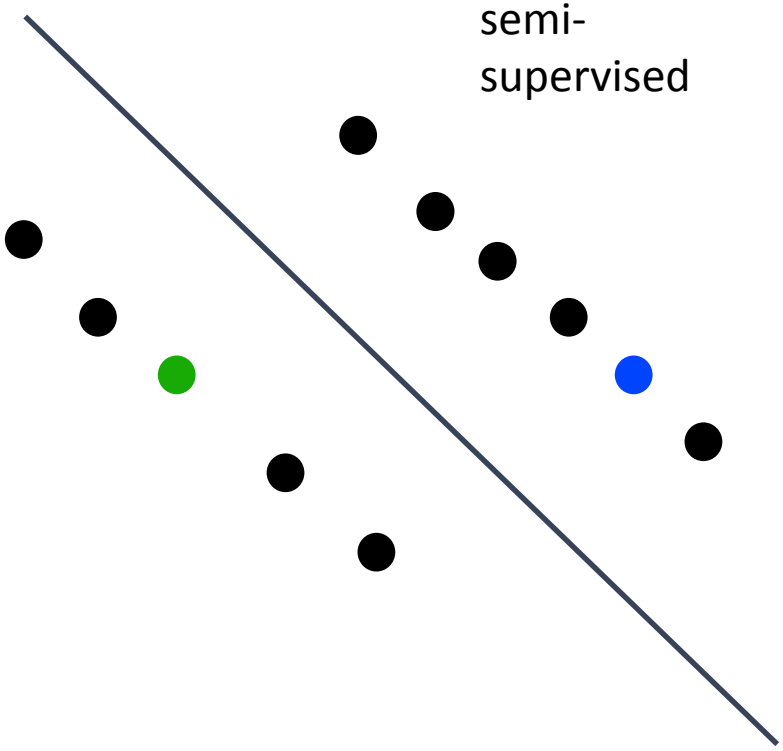
Semi-Supervised Learning

- Hypothesis: P($c|x$) can be more accurately computed using shared structure with P(x)



Semi-Supervised Learning

- Hypothesis: P($c|x$) can be more accurately computed using shared structure with P(x)



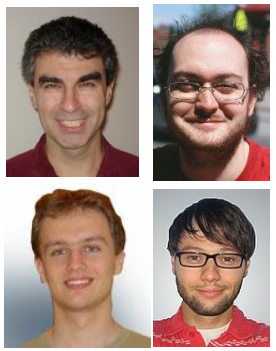
Deep autoencoders

- Alternative to contrastive unsupervised word learning
  - Another is RBMs (Hinton et al. 2006), next section

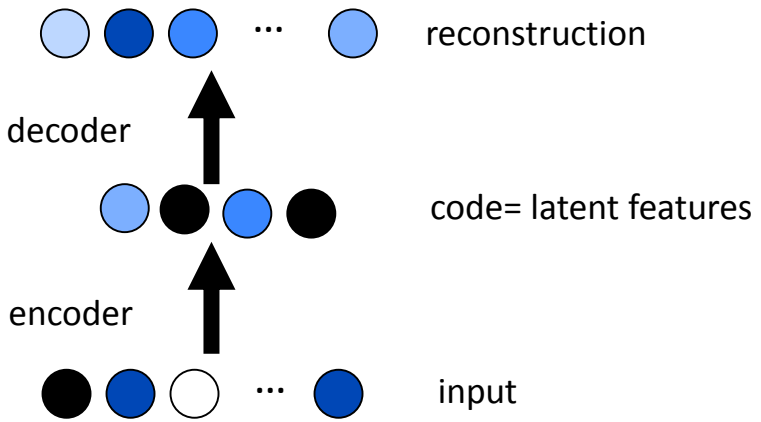
1. Definition, intuition and variants of autoencoders
2. Stacking for deep autoencoders

Auto-Encoders

- Multilayer neural net with target output = input
- Reconstruction=decoder(encoder(input))



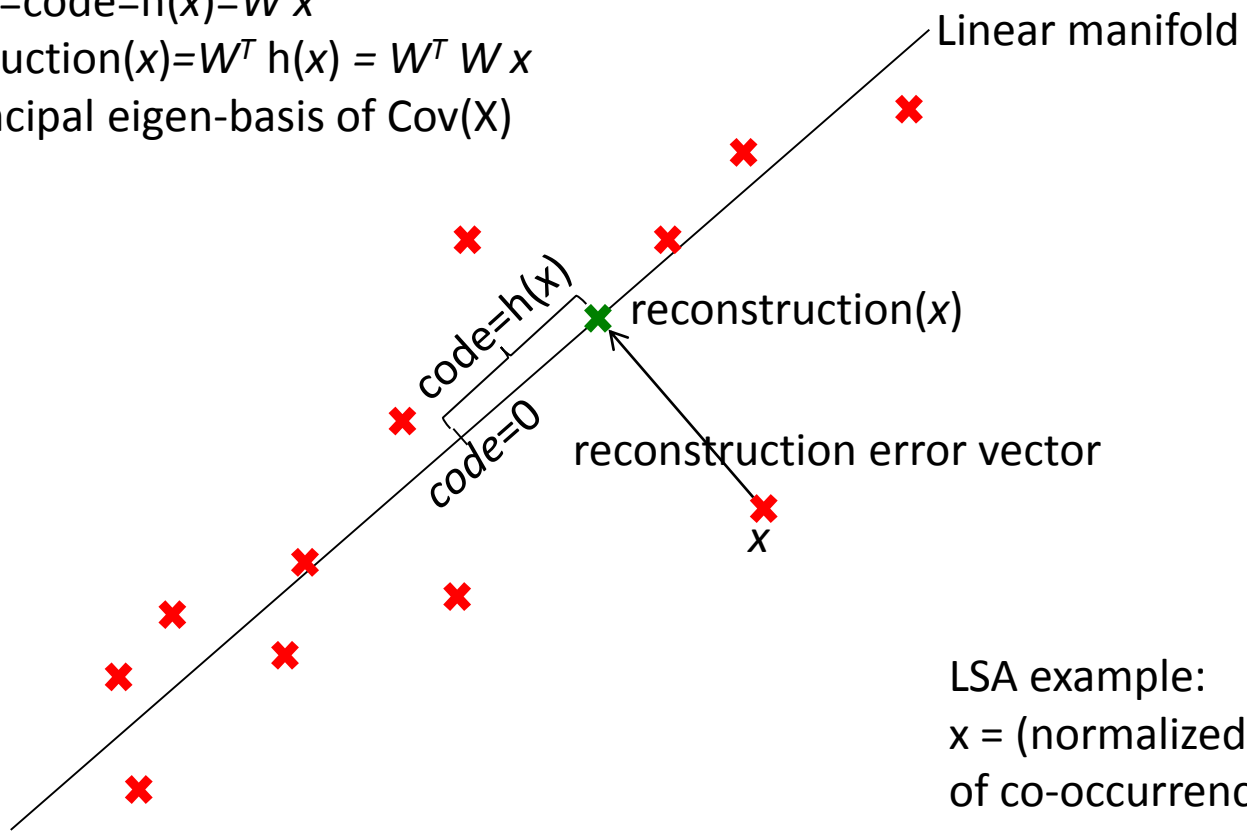
$$
\begin{aligned}
a &= \tanh(Wx + b) \\
x' &= \tanh(W^T a + c) \\
cost &= ||x' - x||^2
\end{aligned}
$$



- Probable inputs have small reconstruction error

PCA = Linear Manifold = Linear Auto-Encoder

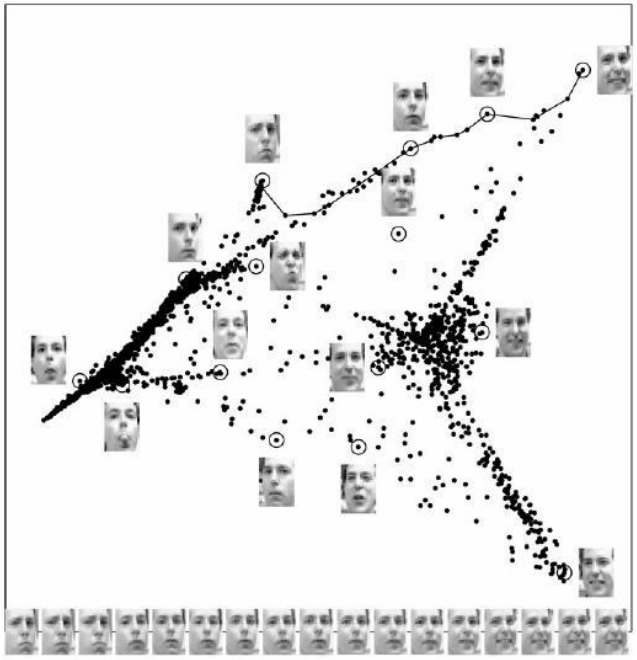
input x, 0-mean
features=code=h(x)=$W\,x$
reconstruction(x)=W^T h(x) = $W^T\,W\,x$
W = principal eigen-basis of Cov(X)

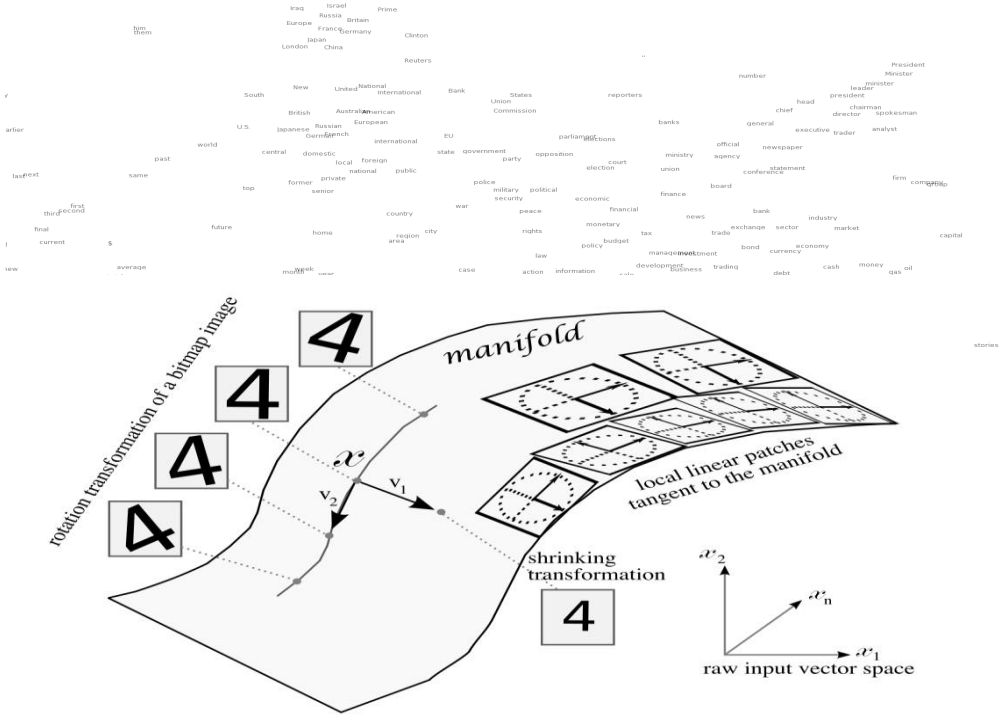


LSA example:
x = (normalized) distribution
of co-occurrence frequencies

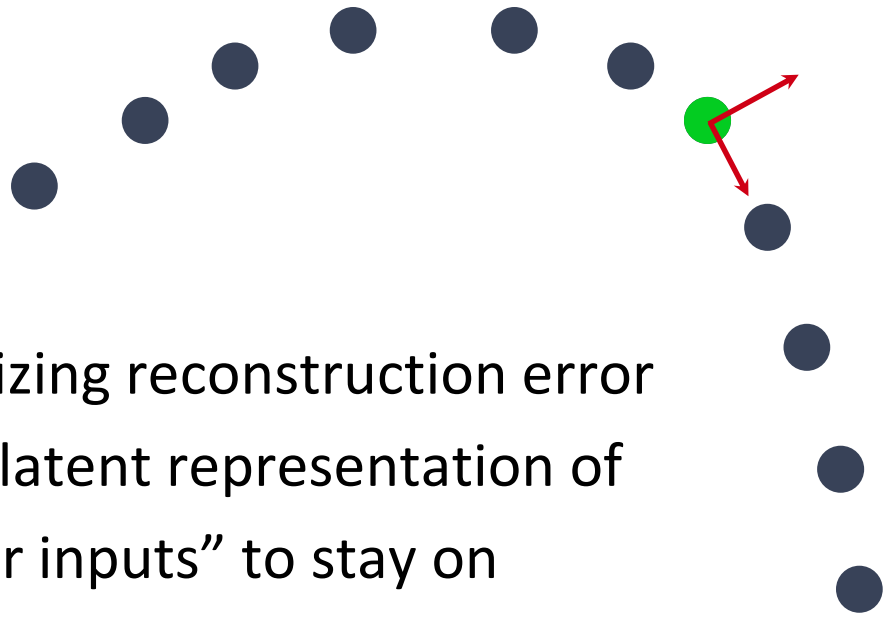
The Manifold Learning Hypothesis

- Examples concentrate near a lower dimensional “manifold” (region of high density where small changes are only allowed in certain directions)





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



Minimizing reconstruction error forces latent representation of "similar inputs" to stay on manifold

Auto-Encoder Variants

- Discrete inputs: cross-entropy or log-likelihood reconstruction criterion (similar to used for discrete targets for MLPs)
- Preventing them to learn the identity everywhere:
  - Undercomplete (eg PCA): bottleneck code smaller than input
  - Sparsity: penalize hidden unit activations so at 0 or near (0.05)

    [Goodfellow et al 2009] $-\rho \log \bar{h}_j - (1-\rho)\log(1-\bar{h}_j)$

    
  - Contractive: force encoder to have small derivatives

    [Rifai et al 2011]

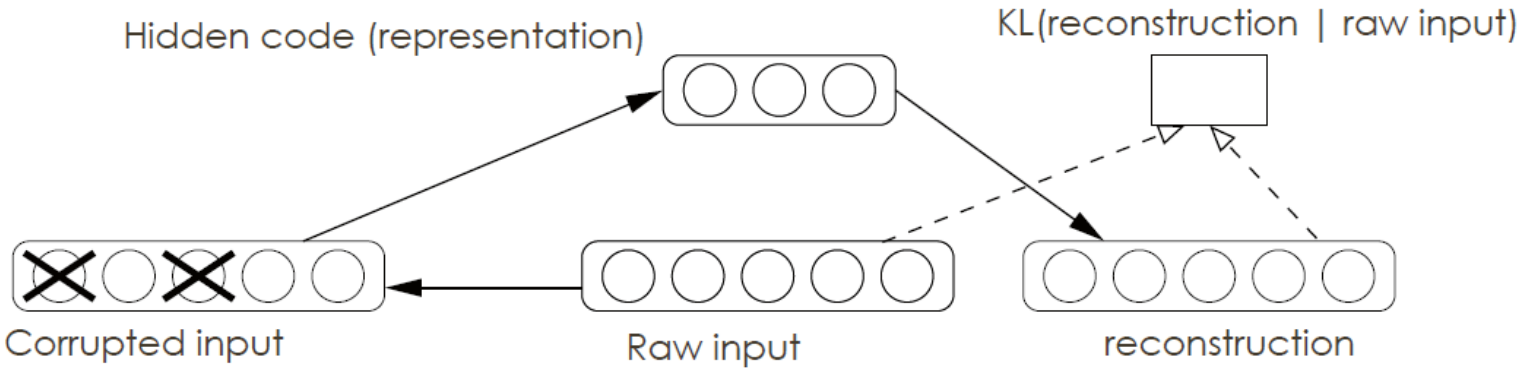
    
  - Denoising: see next slide

    [Vincent et al 2008]

Denoising Autoencoder

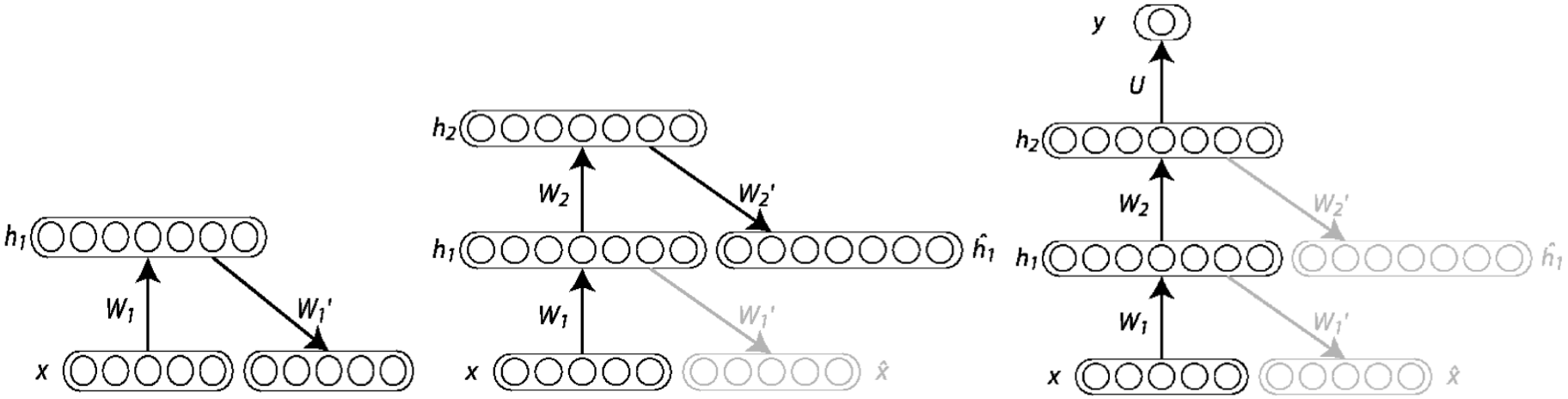
- Corrupt the input
- Reconstruct the uncorrupted input



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

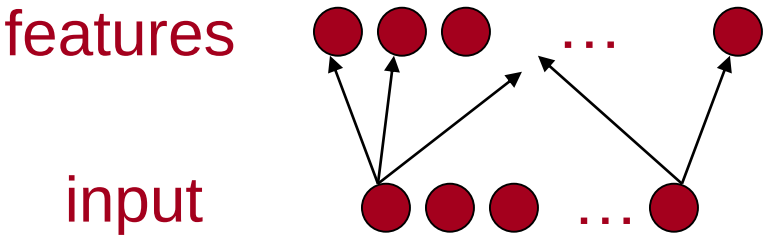
Stacking Auto-Encoders

- Can be stacked successfully (Bengio et al NIPS'2006) to form highly non-linear representations

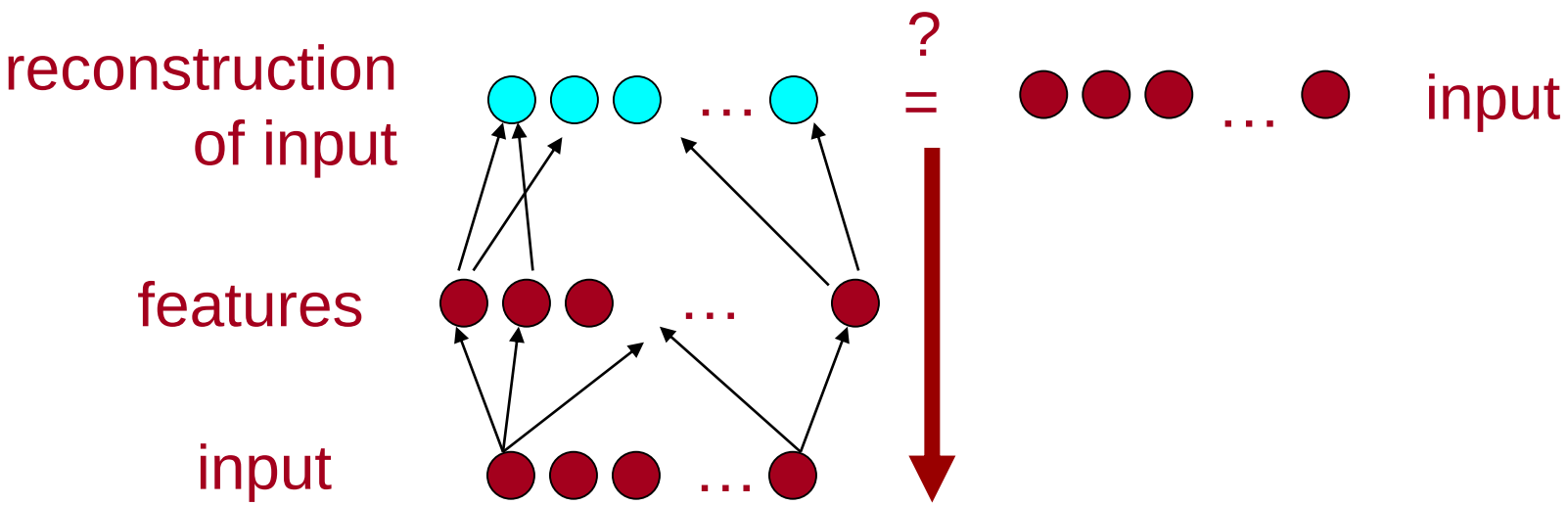


Layer-wise Unsupervised Learning

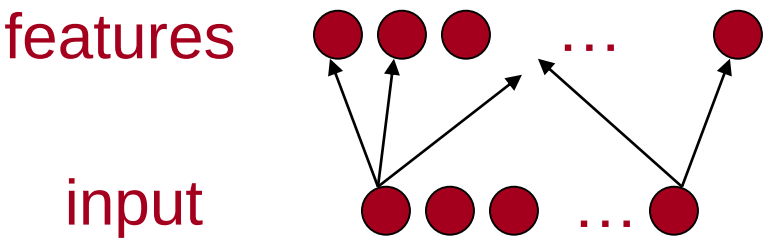
Layer-wise Unsupervised Pre-training



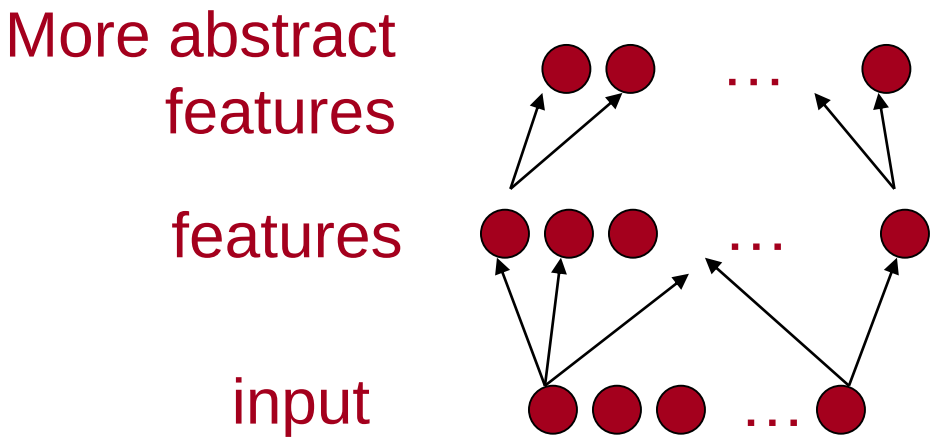
Layer-wise Unsupervised Pre-training



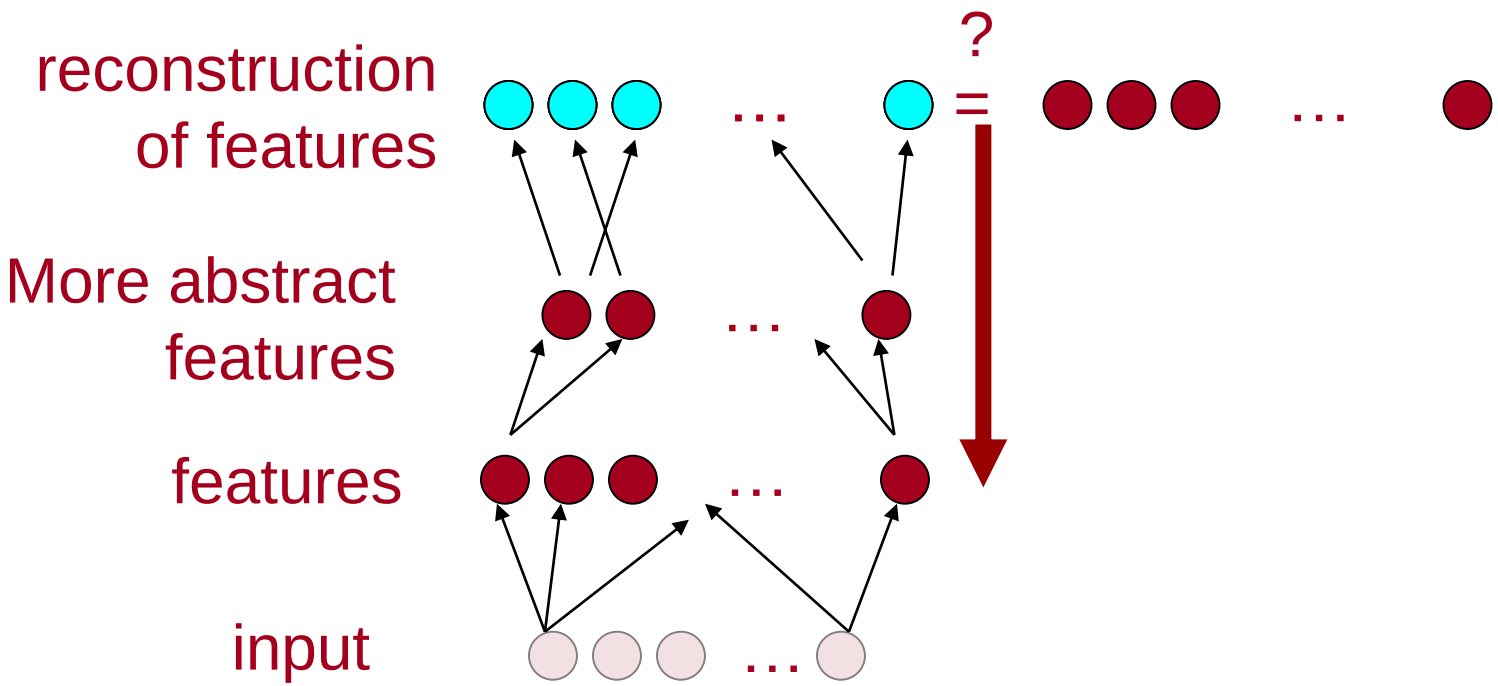
Layer-wise Unsupervised Pre-training



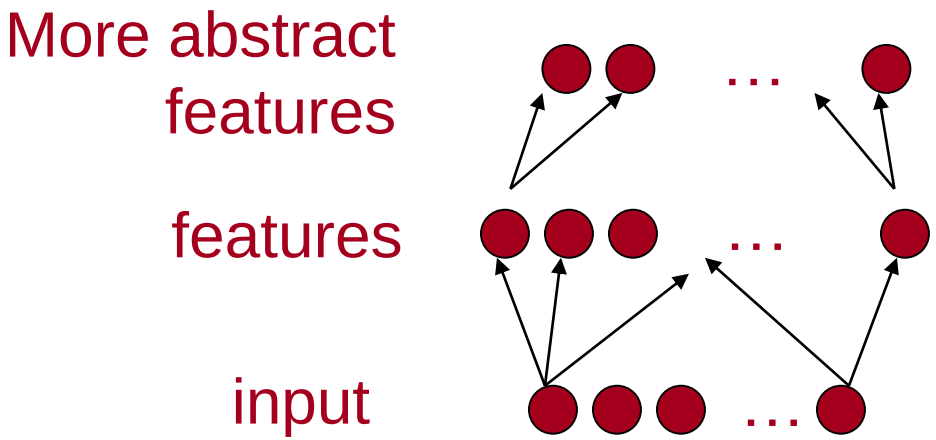
Layer-wise Unsupervised Pre-training



Layer-wise Unsupervised Learning



Layer-wise Unsupervised Pre-training



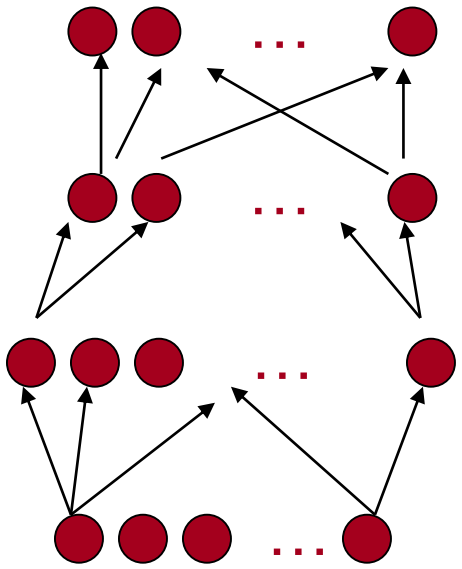
Layer-wise Unsupervised Learning

Even more abstract features

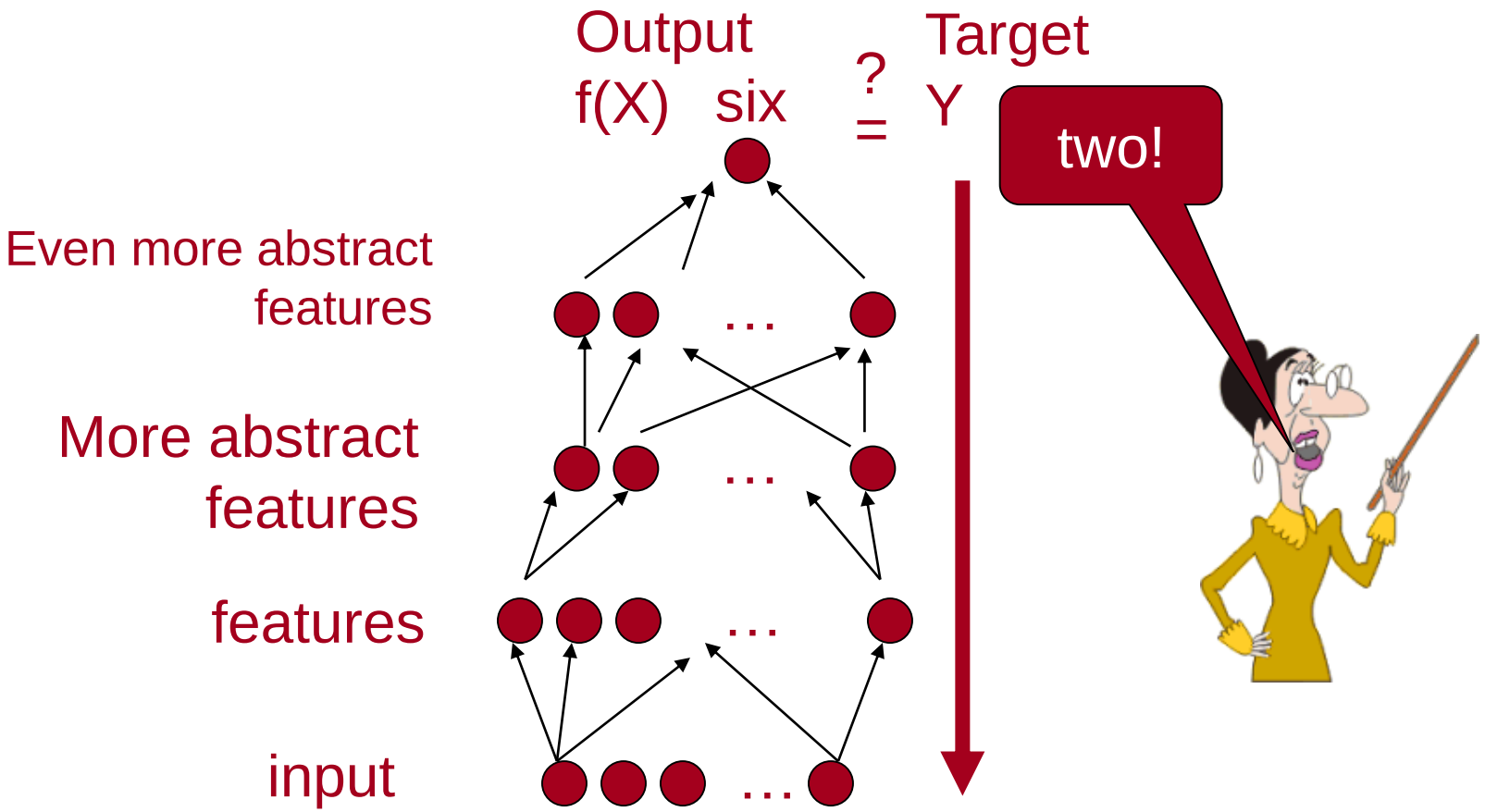
More abstract features

features

input



Supervised Fine-Tuning



Outline

1. Motivation
2. Neural Networks: Feed-forward*, Autoencoders
3. Probabilistic - Directed: PCA, Sparse Coding
4. Probabilistic – Undirected: MRFs and RBMs

Probabilistic Models

- Learn a joint probability distribution over p(x,h)
- Maximize likelihood
- See posterior p(h|x) as the feature representation

Directed Graphical Models: PCA

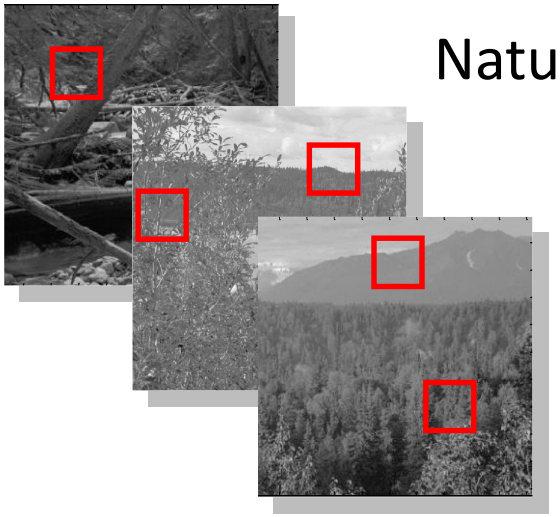
- Factorization of p(x,h) = p(x|h) p(h) = likelihood x prior

- Probabilistic interpretation of PCA:

$$
\begin{aligned}
p(h) &= \mathcal{N}(h; 0, \sigma_h^2 \mathbf{I}) \\
p(x \mid h) &= \mathcal{N}(x; Wh + \mu_x, \sigma_x^2 \mathbf{I})
\end{aligned}
$$

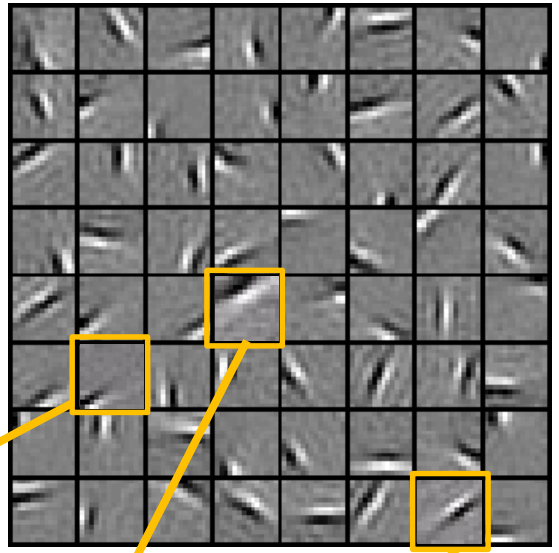
- Get W via iterative maximization of likelihood (i.e. EM)

Sparse Coding: Intuition for images

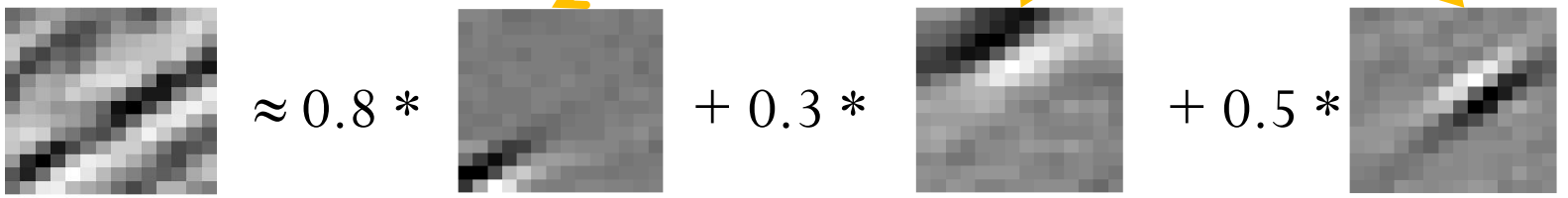


Natural Images

Learned bases:



Test example



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

(feature representation)

Directed Graphical Models: Sparse Coding

- Sparse Coding (non-probabilistic)
  - Training:

$$\min_{D,s(i)} \sum_i ||Ds^{(i)} - x^{(i)}||_2^2 + \lambda ||s^{(i)}||_1$$

$$\text{subject to } ||D^{(j)}||_2^2 = 1, \forall j$$

  - Testing:

$$\min_{s(i)} \sum_i ||Ds^{(i)} - x^{(i)}||_2^2 + \lambda ||s^{(i)}||_1$$

Directed Graphical Models: Sparse Coding

- Sparse Coding (probabilistic)
- Similar to pPCA but with Laplace prior on hidden variables

$$p(h) = \prod_{i}^{d_h} \lambda \exp(-\lambda |h_i|)$$

$$p(x \mid h) = \mathcal{N}(x; Wh + \mu_x, \sigma_x^2 \mathbf{I})$$

- In practice Adam Coates showed that dictionary D can be learned with k-means and encoding (test time) can use soft threshold:

$$\min_{D, s(i)} \sum_i ||Ds^{(i)} - x^{(i)}||_2^2 + \lambda ||s^{(i)}||_1$$

$$\text{subject to } ||D^{(j)}||_2^2 = 1, \forall j$$

$$f_j = \max\left\{0, D^{(j)\top} x - \alpha\right\}$$

Undirected Graphical Models: MRFs and Boltzmann machines

- Generally p(x,h) defined via non-negative clique potentials

$$p(x,h) = \frac{1}{Z_\theta} \prod_i \psi_i(x) \prod_j \eta_j(h) \prod_k \nu_k(x,h)$$

- Special case: Boltzmann machine, energy

$$p(x,h) = \frac{1}{Z_\theta} \exp\left(-\mathcal{E}_\theta(x,h)\right)$$

$$\mathcal{E}_\theta^{\mathrm{BM}}(x,h) = -\frac{1}{2}x^T U x - \frac{1}{2}h^T V h - x^T W h - b^T x - d^T h,$$

- Gives sigmoid conditional for hidden: $P(h_i \mid x, h_{\setminus i}) = \mathrm{sigmoid}\left(\sum_j W_{ji}x_j + \sum_{i' \neq i} V_{ii'}h_{i'} + d_i\right)$
- Totally intractable since single hidden unit requires marginalization over all others:

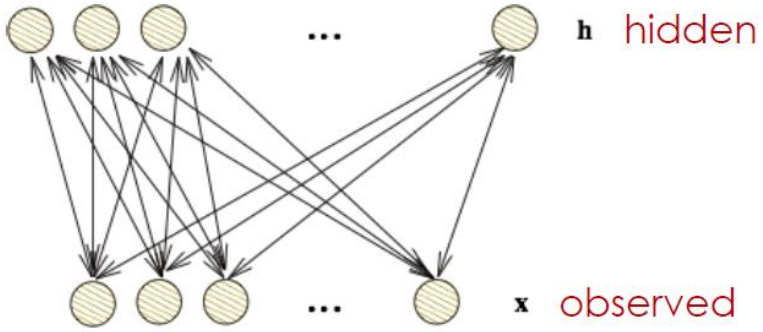
$$P(h_i \mid x) = \sum_{h_1=0}^{h_1=1} \cdots \sum_{h_{i-1}=0}^{h_{i-1}=1} \sum_{h_{i+1}=0}^{h_{i+1}=1} \cdots \sum_{h_{d_h}=0}^{h_{d_h}=1} P(h \mid x)$$

Undirected Graphical Models: Restricted Boltzmann machines (RBMs)

- RBM energy only has visible-hidden connections

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

- Bipartite graph



- Conditionals nicely factorize

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

$$P(h_i = 1 \mid x) = \text{sigmoid}\left(\sum_j W_{ji} x_j + d_i\right)$$

Undirected Graphical Models: Restricted Boltzmann machines (RBMs)

- Detail: Factorization

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

$$
P(h_i = 1|x) = \frac{e^{c_i + W_i x}}{1 + e^{c_i + W_i x}} = \frac{1}{1 + e^{-(c_i + W_i x)}} = \frac{1}{1 + e^{-z_i}} = \sigma(z_i) = f(z_i)
$$

Undirected Graphical Models: Restricted Boltzmann machines (RBMs)

- Detail: Training RBMs
- Maximize marginal $P(x) = \sum_h \frac{e^{-En(x,h)}}{Z}$ $\quad Z = \sum_{\tilde{x}} \sum_h e^{-En(\tilde{x},h)}$

$$
\begin{aligned}
\frac{\partial \log P(x)}{\partial \theta} &= \frac{\partial}{\partial \theta} \log \left(\sum_h e^{-En(x,h)} \right) - \frac{\partial}{\partial \theta} \log \left(\sum_{\tilde{x}} \sum_h e^{-En(x,h)} \right) \\
&= -\frac{1}{\sum_h e^{-En(x,h)}} \sum_h e^{-En(x,h)} \frac{\partial}{\partial \theta} En(x,h) + \frac{1}{\sum_{\tilde{x}} \sum_h e^{-En(x,h)}} \sum_{\tilde{x}} \sum_h e^{-En(x,h)} \frac{\partial}{\partial \theta} En(\tilde{x},h) \\
&= -\sum_h \frac{e^{-En(x,h)}}{\sum_h e^{-En(x,h)}} \frac{\partial}{\partial \theta} En(x,h) + \sum_{\tilde{x}} \sum_h \frac{e^{-En(\tilde{x},h)}}{\sum_{\tilde{x}} \sum_h e^{-En(\tilde{x},h)}} \frac{\partial}{\partial \theta} En(\tilde{x},h) \\
&= -\sum_h P(h|x) \frac{\partial}{\partial \theta} En(x,h) + \sum_{\tilde{x}} \sum_h P(\tilde{x},h) \frac{\partial}{\partial \theta} En(\tilde{x},h) \\
&= -E_{\hat{P}(h|x)} \left[\frac{\partial}{\partial \theta} En(x,h) \right] + E_{P(\tilde{x},h)} \left[\frac{\partial}{\partial \theta} En(\tilde{x},h) \right].
\end{aligned}
$$

$$\frac{\partial}{\partial W_{ij}} En(x,h) = -h_i x_j.$$

Undirected Graphical Models: Restricted Boltzmann machines (RBMs)

- Detail: Training RBMs

$$-E_{\hat{P}(h|x)}\left[\frac{\partial}{\partial\theta}En(x,h)\right]+E_{P(\tilde{x},h)}\left[\frac{\partial}{\partial\theta}En(\tilde{x},h)\right].$$

- Conditional of empirical distributions is easy $\frac{\partial}{\partial W_{ij}}En(x,h)=-h_i x_j.$
- Expected value wrt joint distribution tricky!
- Use average over samples from MCMC approximation
- MAGIC: Contrastive Divergence, Hinton 2006: $x^{(1)}\sim\hat{P}(x),h^{(1)}\sim P(h|x^{(1)}),x^{(2)}\sim P(x|h^{(1)}$
  - Instead of many samples, use only 1
  - Instead of waiting for burn in, take just one step & start with data sample