School of Computer Science
Carnegie Mellon

10-708 Probabilistic Graphical Models

Deep Neural Networks and Graphical Models

Readings:
Deng (2013)
Bengio (2009)
Hinton (2010)

Matt Gormley
Lecture 26
April 18, 2016

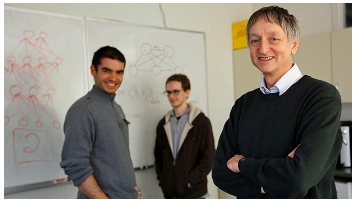
Reminders

- HW4: due April 27
- Project presentations: April 29
  - Location: Baker Hall A51
  - Session 1: 8:30 - 12:30 (4 hrs)
  - Lunch break: 12:30 - 1:30 (1 hr)
  - Session 2: 1:30 - 5:00 (3.5 hrs)

Motivation

Why is everyone talking about Deep Learning?

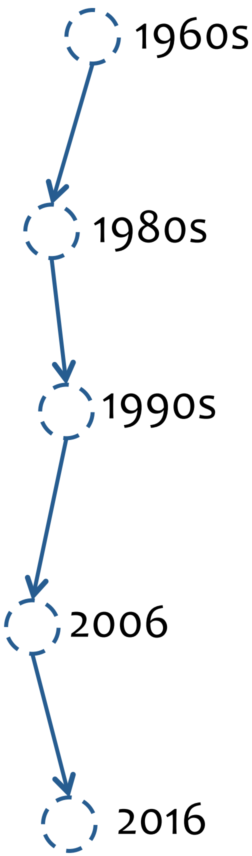
- Because a lot of money is invested in it…
  - DeepMind: Acquired by Google for **$400 million**
  - DNNResearch: **Three person startup** (including Geoff Hinton) acquired by Google for unknown price tag
  - Enlitic, Ersatz, MetaMind, Nervana, Skylab: Deep Learning startups commanding **millions of VC dollars**



- Because it made the **front page** of the New York Times

Motivation

Why is everyone talking about Deep Learning?



Deep learning:

- Has won numerous pattern recognition competitions
- Does so with minimal feature engineering

This wasn't always the case!

Since 1980s: Form of models hasn't changed much, but lots of new tricks…

- More hidden units
- Better (online) optimization
- New nonlinear functions (ReLUs)
- Faster computers (CPUs and GPUs)

Background

A Recipe for Machine Learning

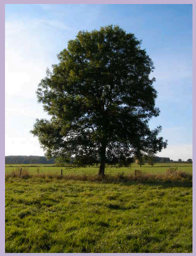
1. Given training data:

$$\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^N$$

Face

Face

Not a face



2. Choose each of these:
   – Decision function

$$\hat{\boldsymbol{y}} = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$$

   – Loss function

$$\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}_i) \in \mathbb{R}$$

Examples: Linear regression, Logistic regression, Neural Network

Examples: Mean-squared error, Cross Entropy

Background

A Recipe for Machine Learning

1. Given training data:

$$\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^N$$

2. Choose each of these:
   - Decision function

$$\hat{\boldsymbol{y}} = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$$

   - Loss function

$$\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}_i) \in \mathbb{R}$$

3. Define goal:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^N \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

4. Train with SGD:
(take small steps opposite the gradient)

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

Gradients

Backpropagation can compute this gradient!

And it's a **special case of a more general algorithm** called reverse-mode automatic differentiation that can compute the gradient of any differentiable function efficiently!

Background

1. Given training dat

$$\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^N$$

2. Choose each of t
   - Decision functio

$$\hat{y} = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$$

   - Loss function

$$\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}_i) \in \mathbb{R}$$

A Recipe for

$\boldsymbol{y}_i)$

opp site the gradient)

$$\boldsymbol{\theta}^{(t} \quad {}^{(t)} - \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

Goals for Today's Lecture

1. Explore a **new class of decision functions** (Deep Nets)
2. Consider **variants of this recipe** for training

A Recipe for

2. Choose each of these:
   - Decision function

$$\hat{y} = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$$

   - Loss function

$$\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}_i) \in \mathbb{R}$$

4. Train with SGD:
(take small steps opposite the gradient)

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

Outline

- **Motivation**
- **Deep Neural Networks (DNNs)**
  - Background: Decision functions
  - Background: Neural Networks
  - Three ideas for training a DNN
  - Experiments: MNIST digit classification
- **Deep Belief Networks (DBNs)**
  - Sigmoid Belief Network
  - Contrastive Divergence learning
  - Restricted Boltzman Machines (RBMs)
  - RBMs as infinitely deep Sigmoid Belief Nets
  - Learning DBNs
- **Deep Boltzman Machines (DBMs)**
  - Boltzman Machines
  - Learning Boltzman Machines
  - Learning DBMs

Outline

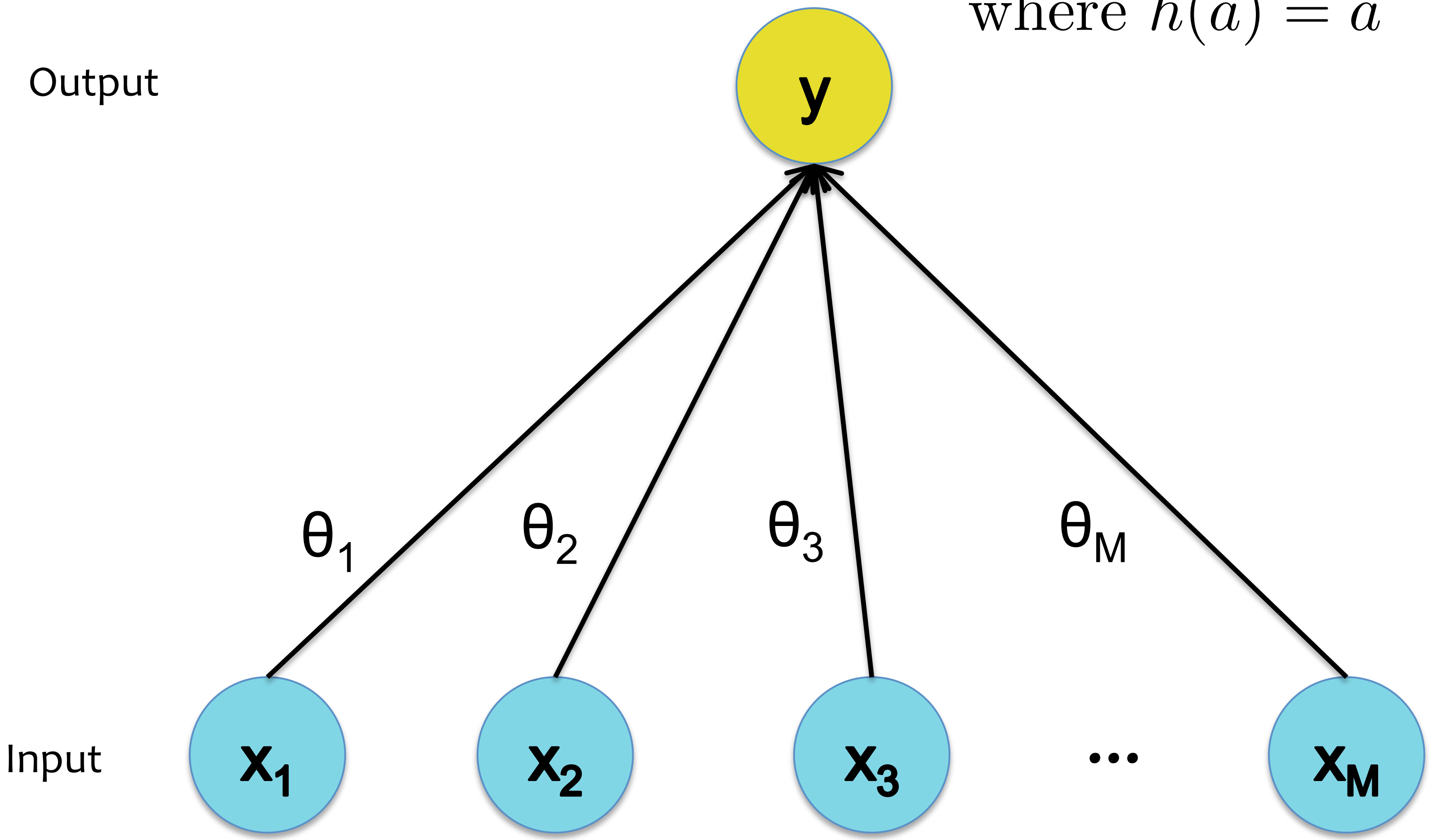
- **Motivation**
- **Deep Neural Networks (DNNs)**
  - Background: Decision functions
  - Background: Neural Networks
  - Three ideas for training a DNN
  - Experiments: MNIST digit classification
- **Deep Belief Networks (DBNs)**
  - Sigmoid Belief Network
  - Contrastive Divergence learning
  - Restricted Boltzman Machines (RBMs)
  - RBMs as infinitely deep Sigmoid Belief Nets
  - Learning DBNs
- **Deep Boltzman Machines (DBMs)**
  - Boltzman Machines
  - Learning Boltzman Machines
  - Learning DBMs

Decision Functions

Linear Regression

$$y = f_{\boldsymbol{\theta}}(\boldsymbol{x}) = h(\boldsymbol{\theta} \cdot \boldsymbol{x})$$

$$\text{where } h(a) = a$$

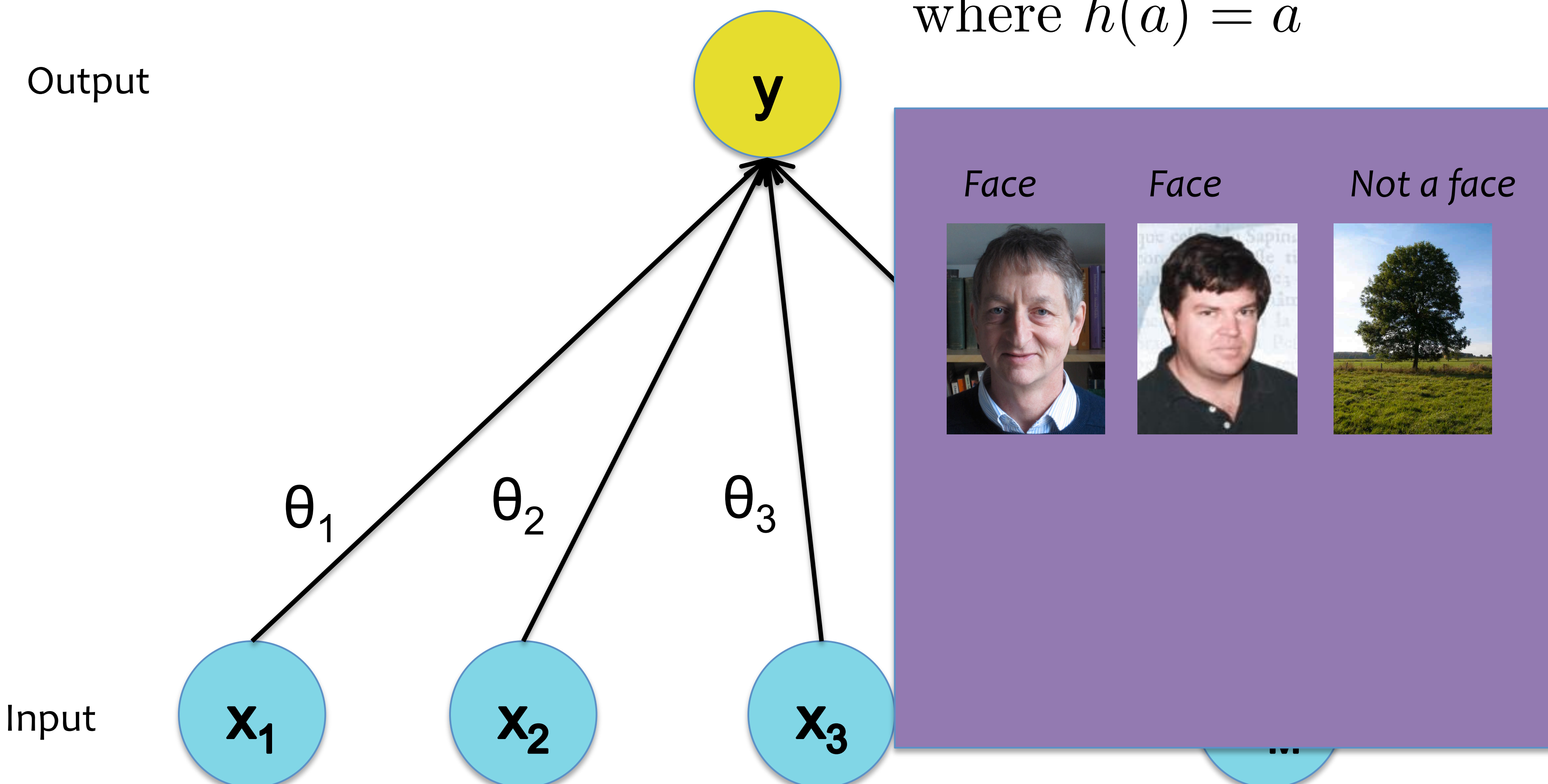


Decision Functions

Linear Regression

$$y = f_{\boldsymbol{\theta}}(\boldsymbol{x}) = h(\boldsymbol{\theta} \cdot \boldsymbol{x})$$

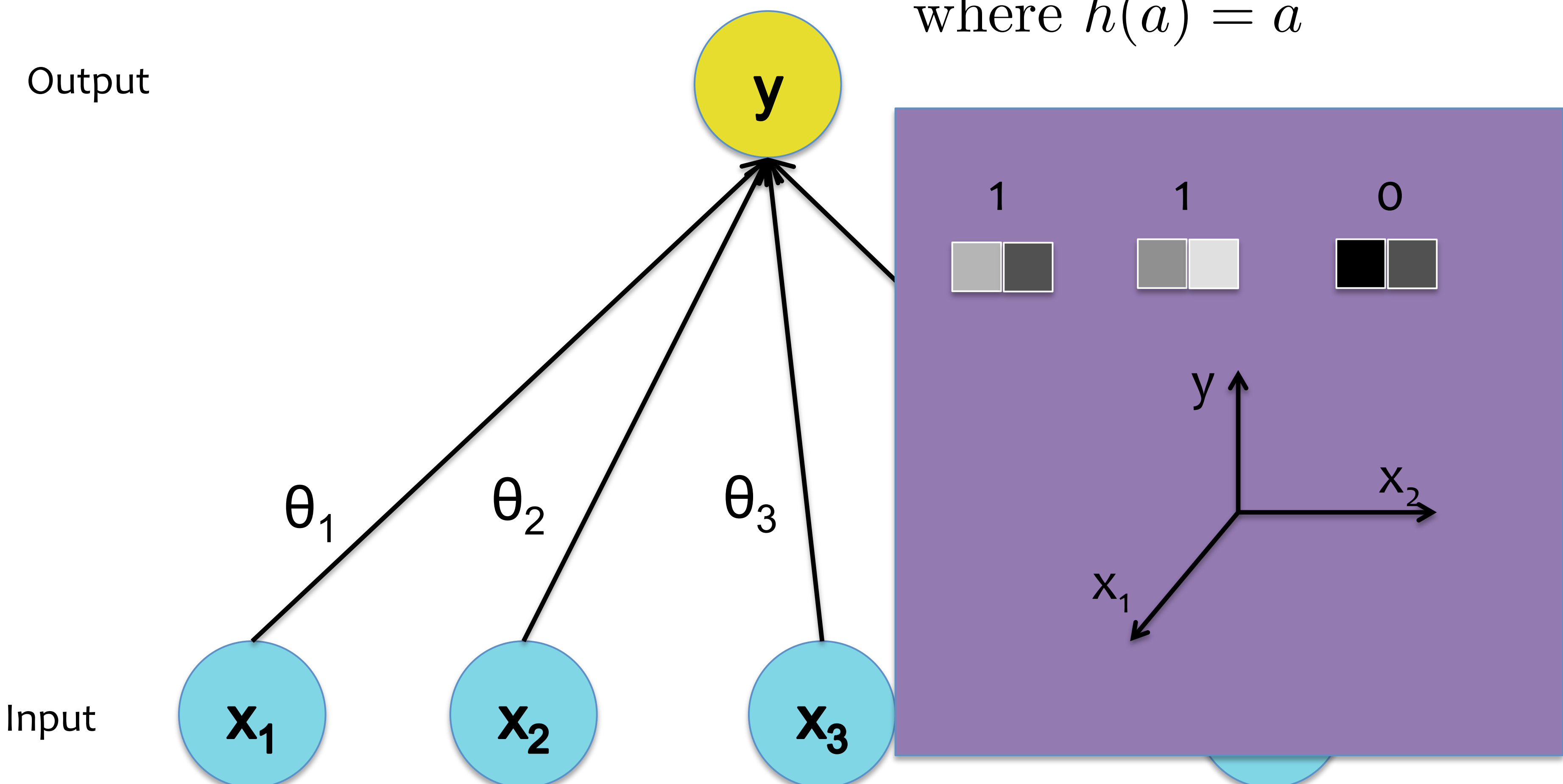
$$\text{where } h(a) = a$$



Decision Functions

Linear Regression

$$y = f_{\boldsymbol{\theta}}(\boldsymbol{x}) = h(\boldsymbol{\theta} \cdot \boldsymbol{x})$$
$$\text{where } h(a) = a$$

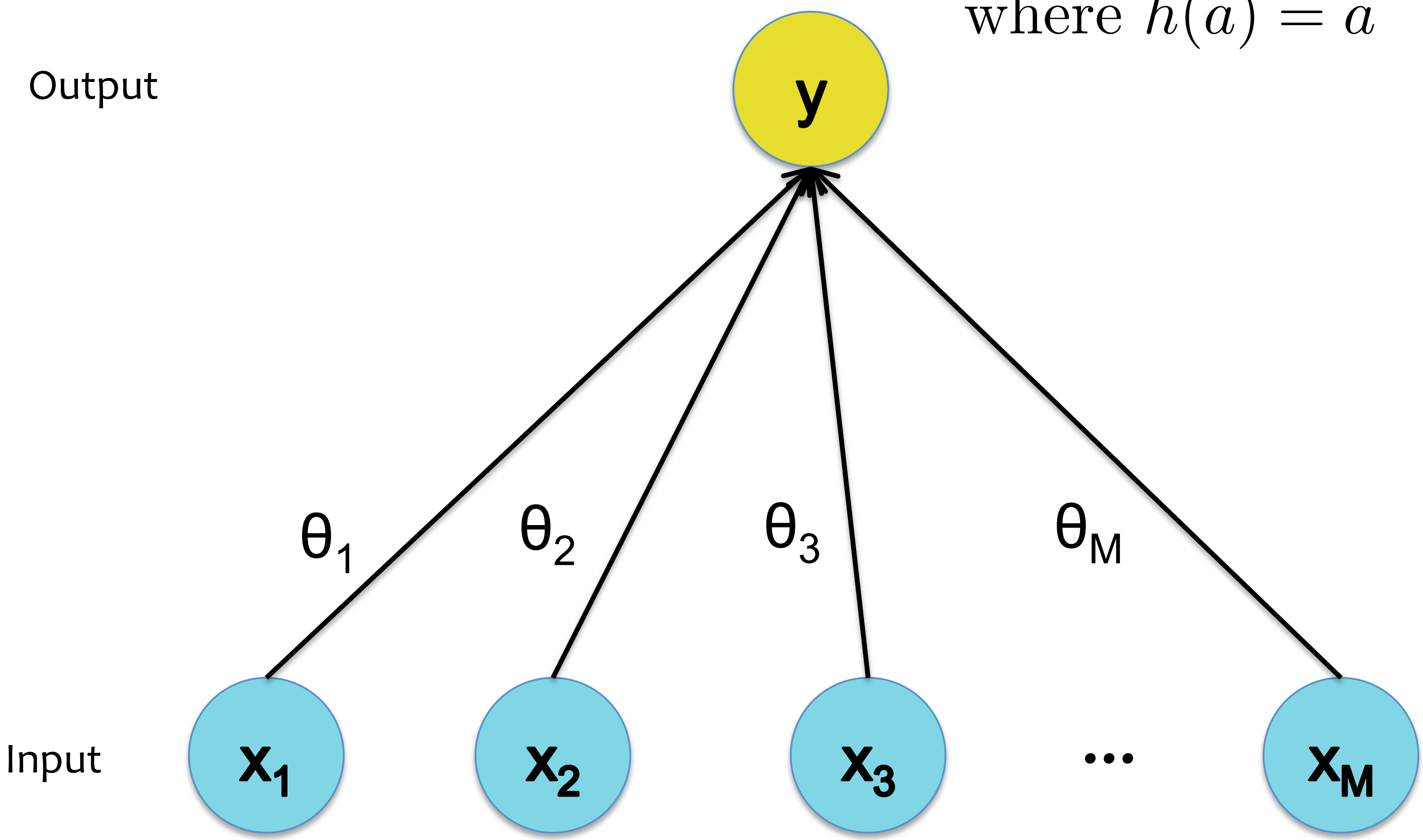


Decision Functions

Linear Regression

$$y = f_{\boldsymbol{\theta}}(\boldsymbol{x}) = h(\boldsymbol{\theta} \cdot \boldsymbol{x})$$

$$\text{where } h(a) = a$$

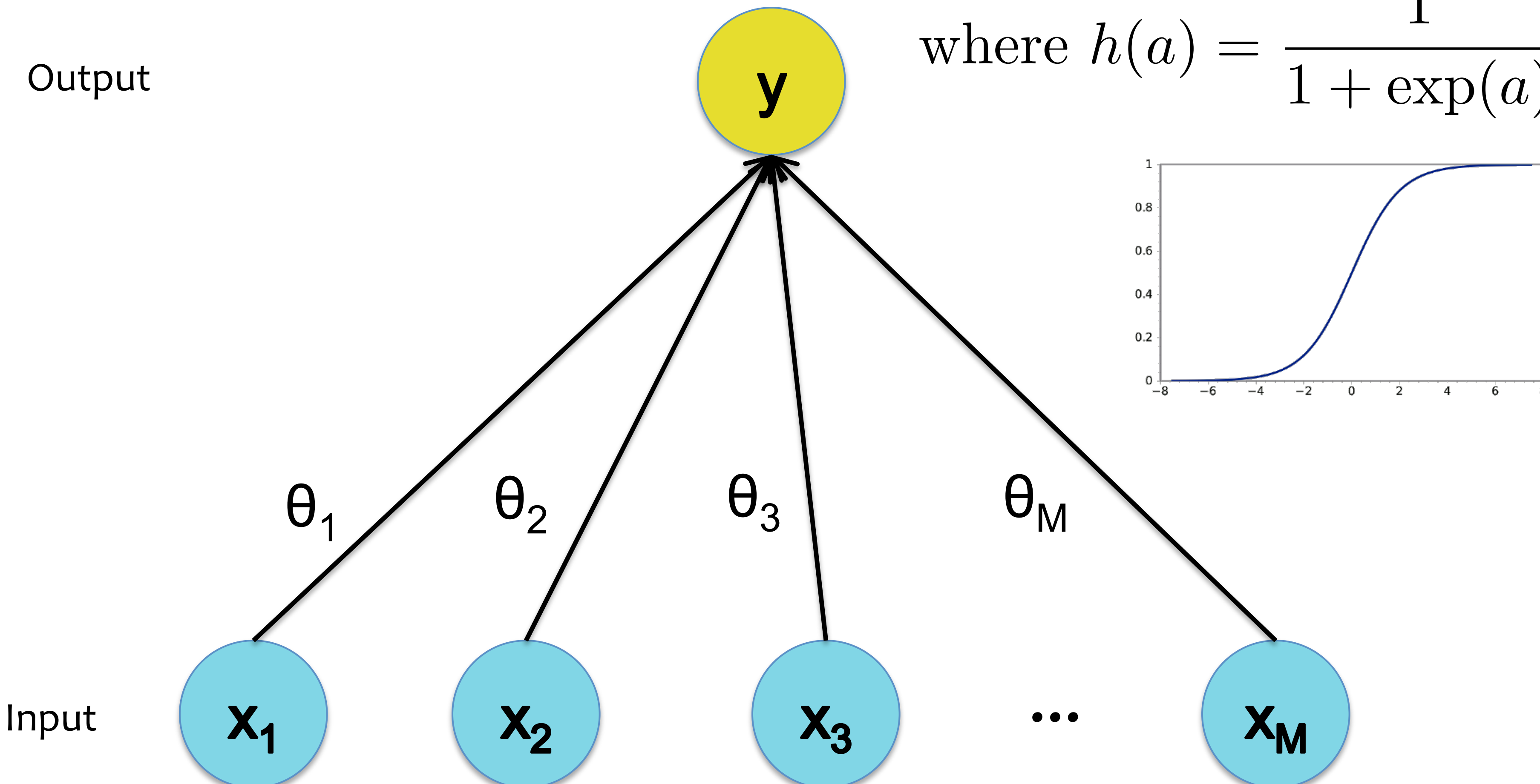


Decision Functions

Logistic Regression

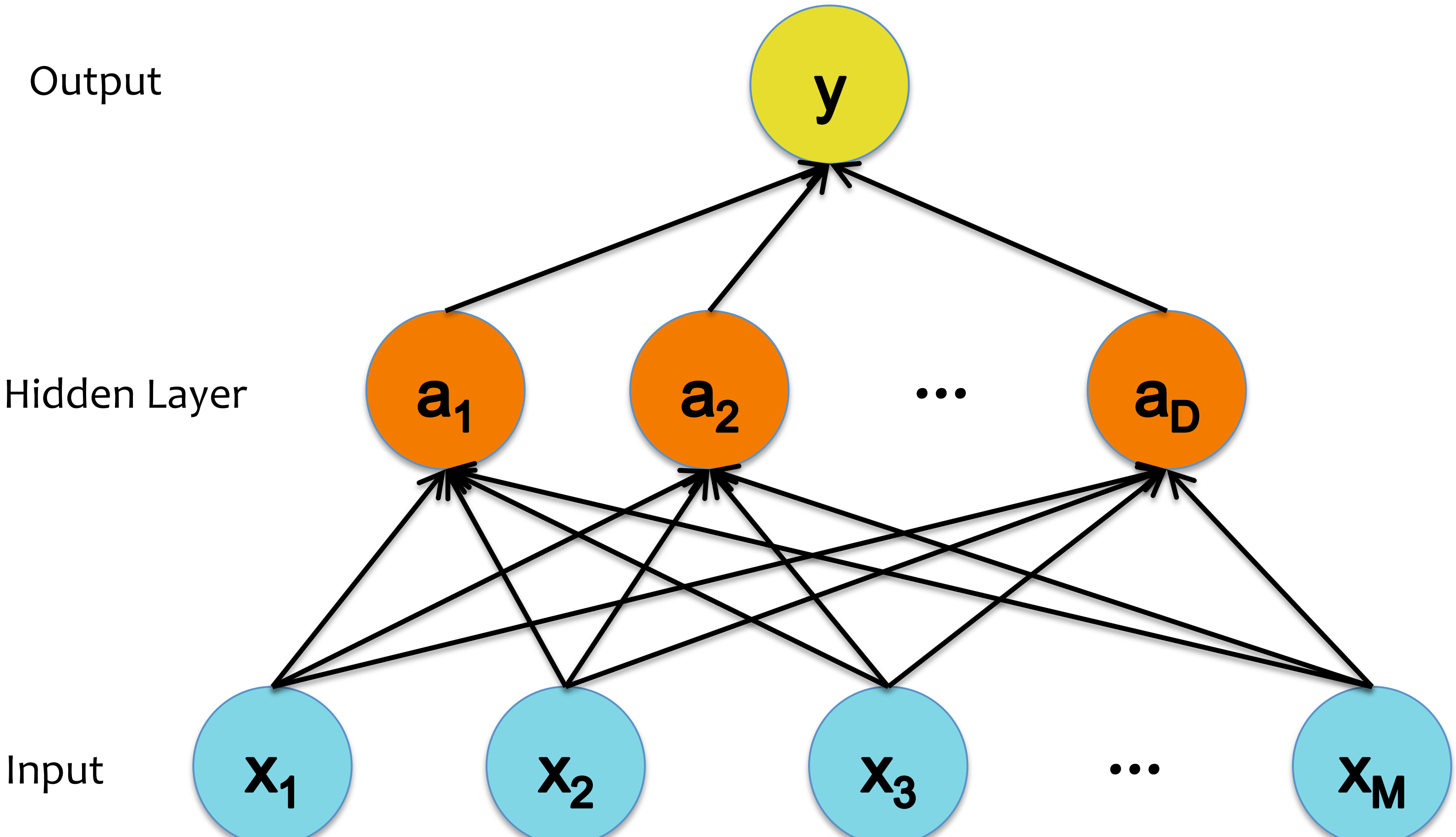
$$y = f_{\boldsymbol{\theta}}(\boldsymbol{x}) = h(\boldsymbol{\theta} \cdot \boldsymbol{x})$$

$$\text{where } h(a) = \frac{1}{1 + \exp(a)}$$

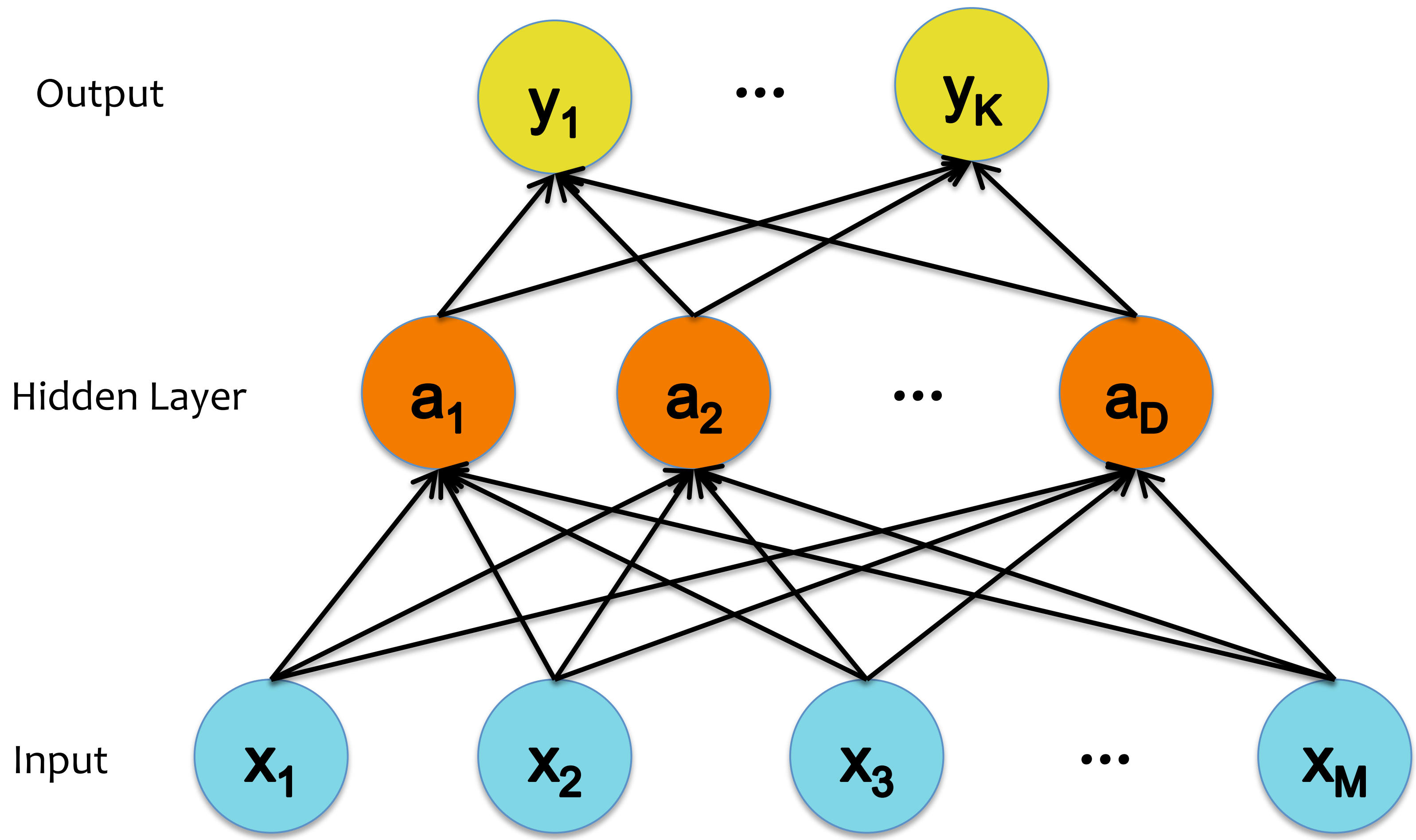


Decision Functions

Neural Network



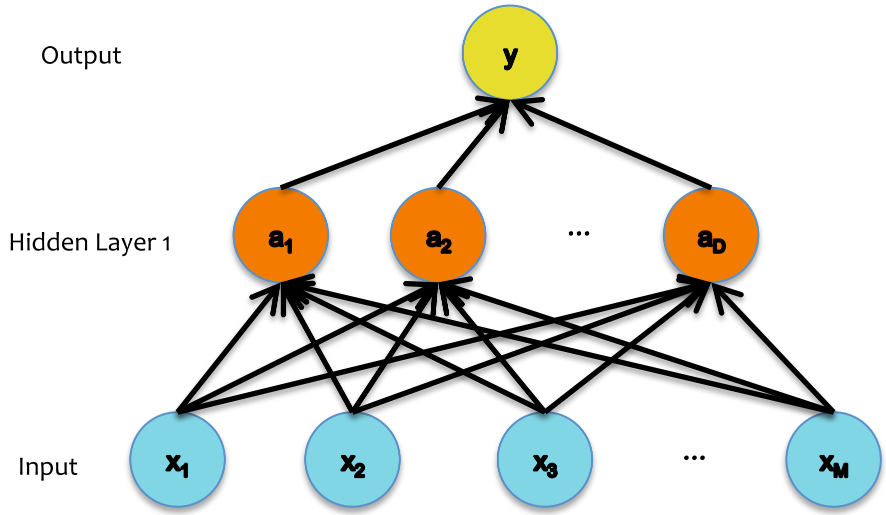
Multi-Class Output



Decision Functions

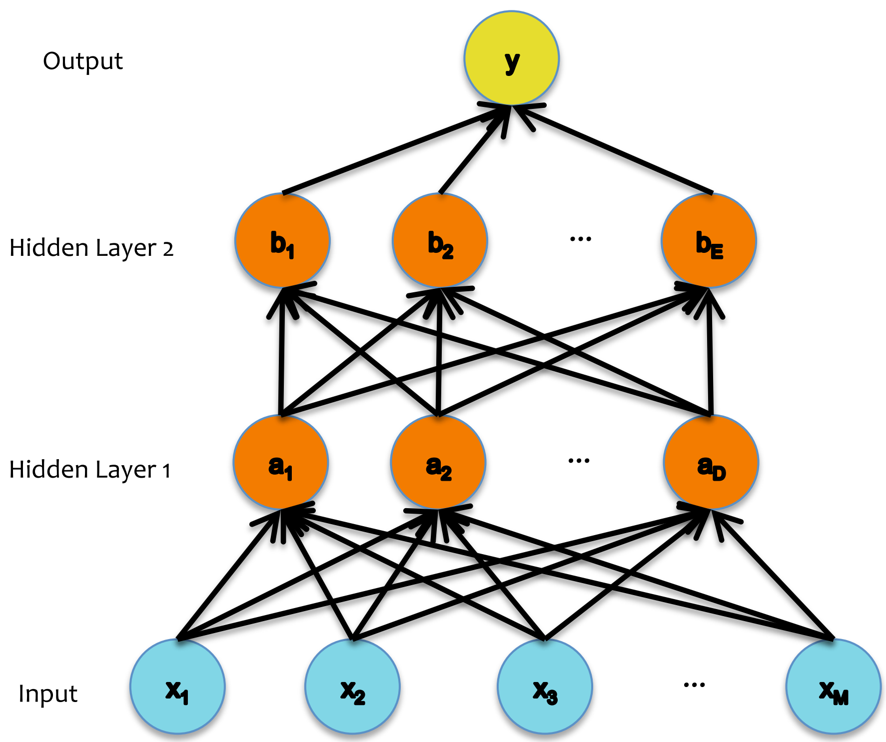
Deeper Networks

This lecture:



Deeper Networks

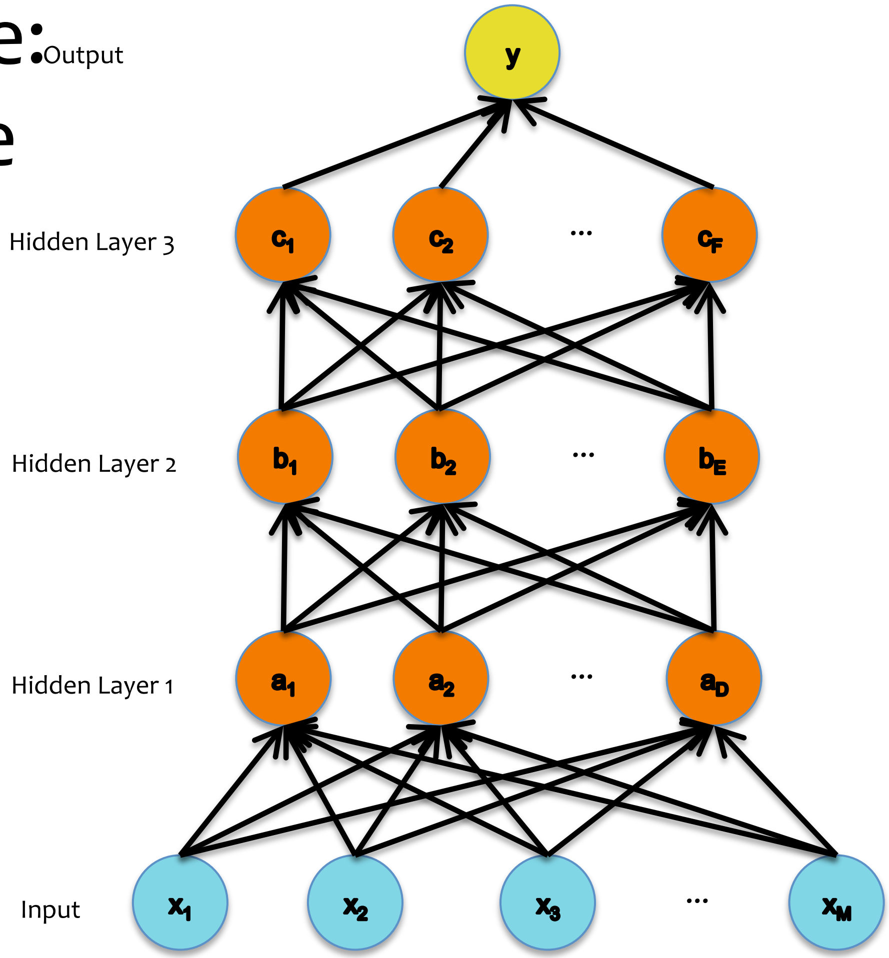
This lecture:



Decision Functions

Deeper Networks

This lecture: Making the neural networks deeper



Decision Functions

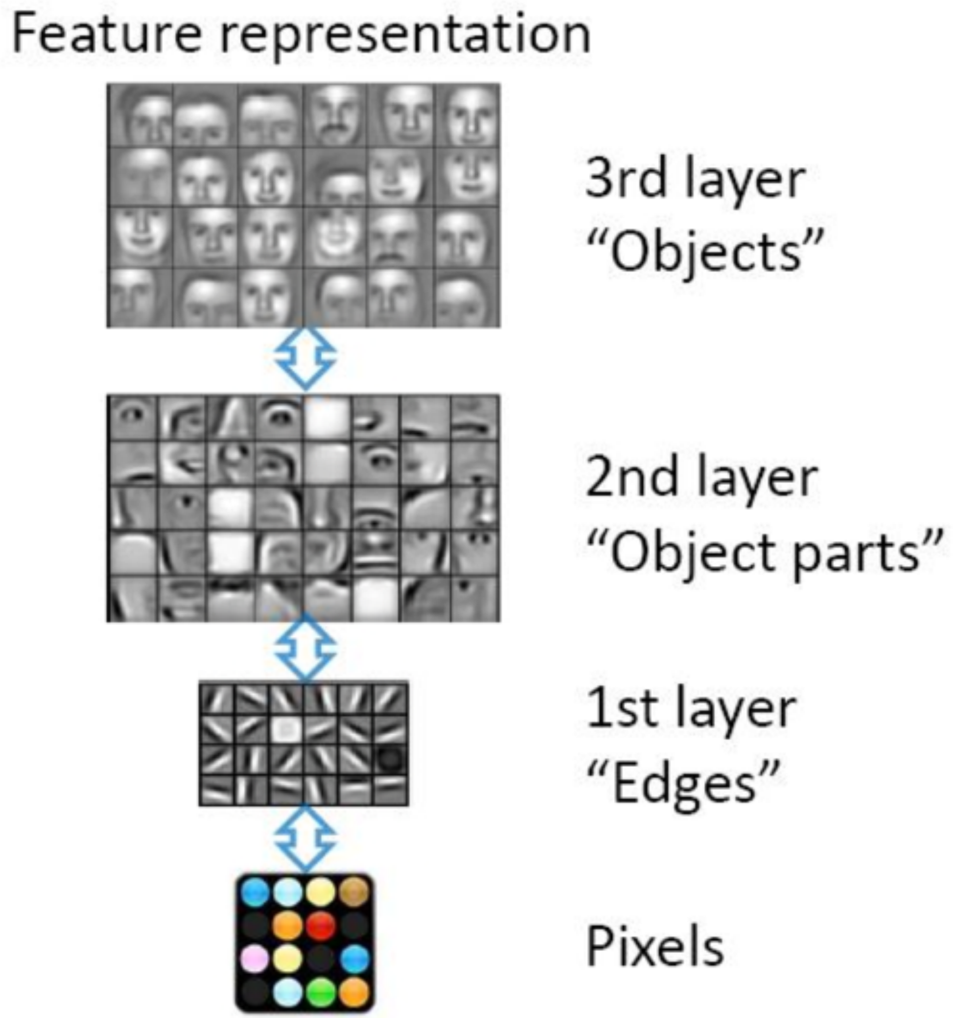
Why go Deep?

Neural Nets (One Hidden Layer)	Deep Networks (Two or more Hidden Layers)
• Already universal function approximators	• Can be representationally efficient • Fewer computational units for the same function
• Can represent non-linear combinations of the input features	• Might allow for a hierarchy • Allows non-local generalizations
• Work well	• Have been shown to work even better (vision, audio, NLP, etc.)!

Slide adapted from Honglak Lee (NIPS 2010)

Different Levels of Abstraction

- We don’t know the “right” levels of abstraction
- So let the model figure it out!



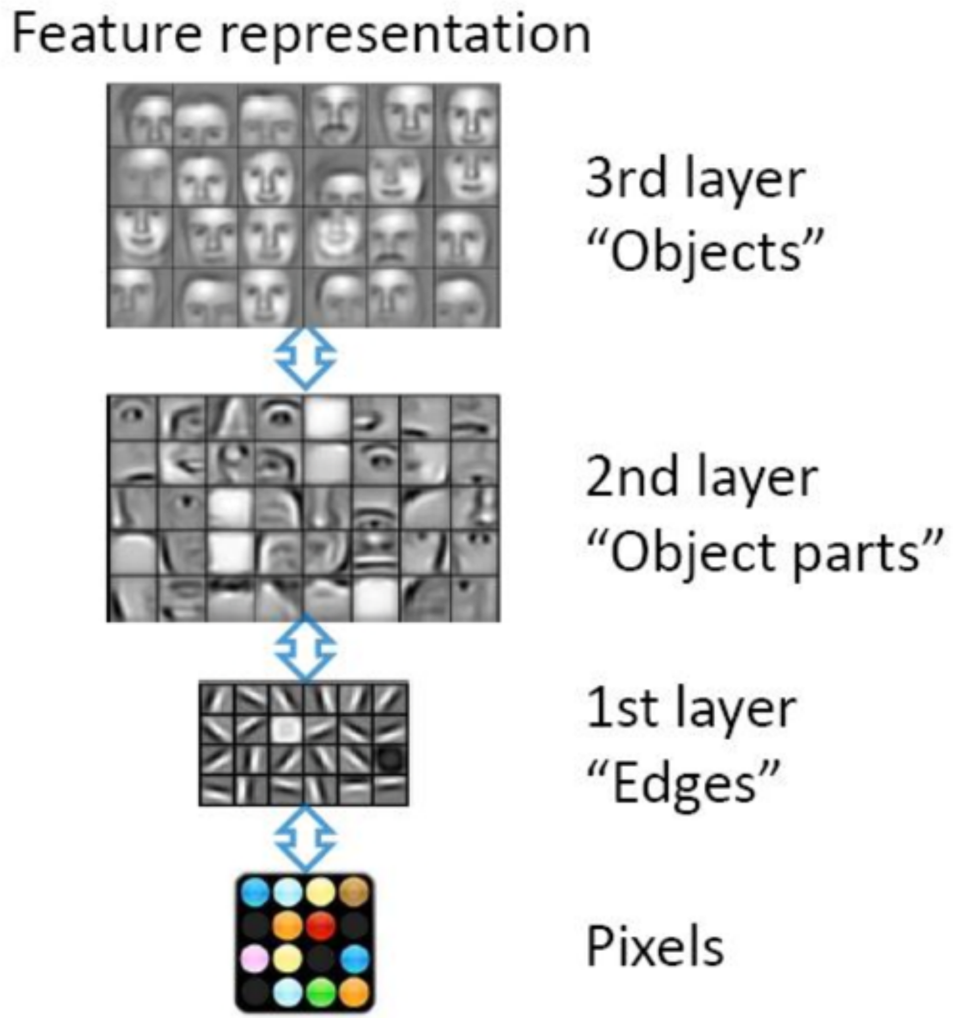
Example from Honglak Lee (NIPS 2010)

Decision Functions

Different Levels of Abstraction

Face Recognition:

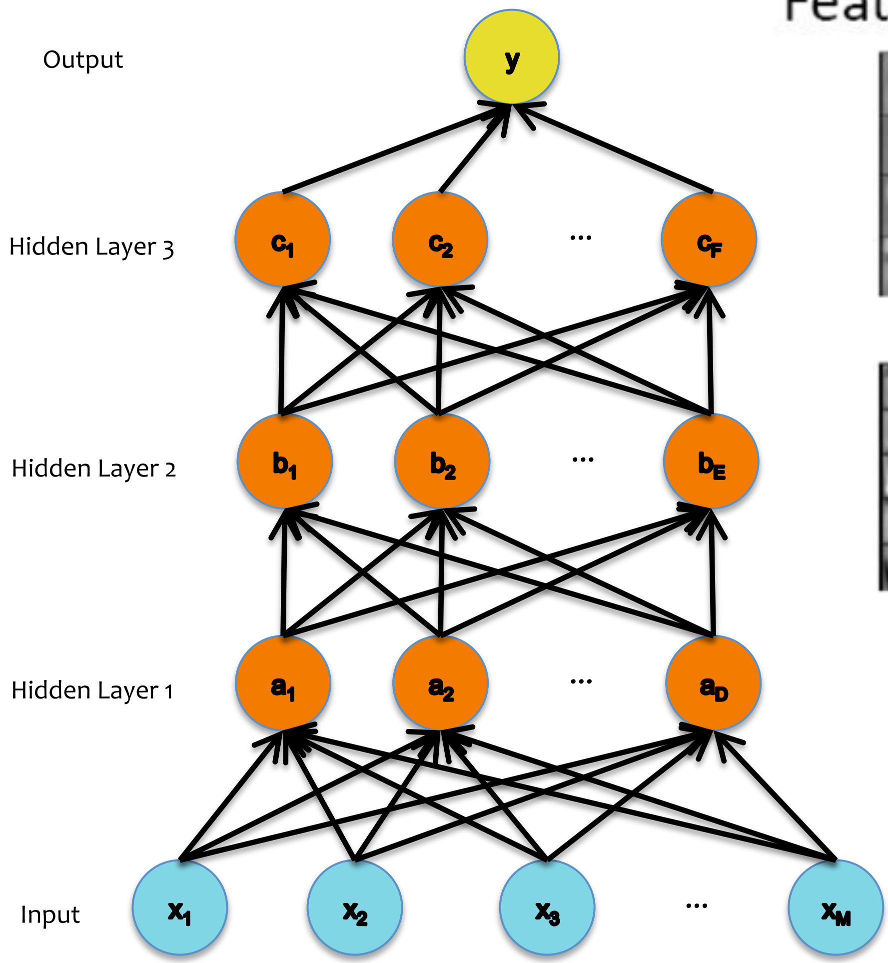
- Deep Network can build up increasingly higher levels of abstraction
- Lines, parts, regions

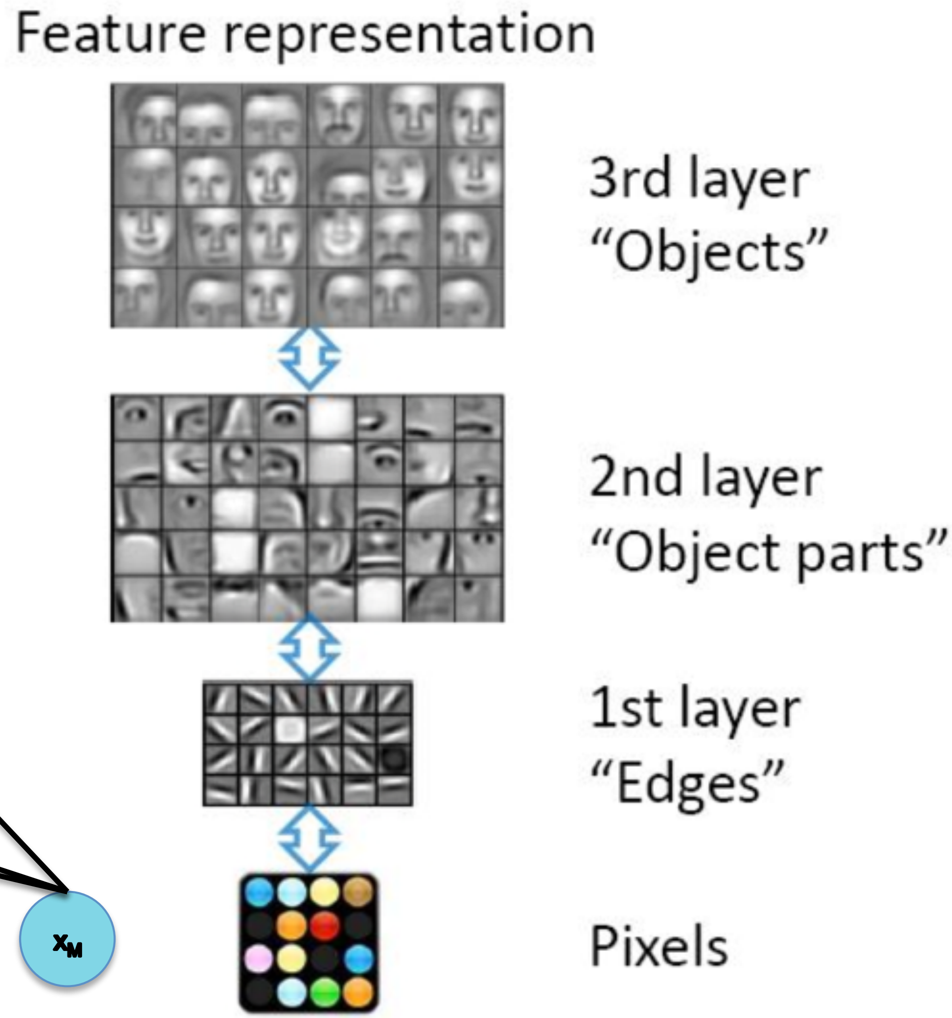


Example from Honglak Lee (NIPS 2010)

Decision Functions

Different Levels of Abstraction





Example from Honglak Lee (NIPS 2010)

Goals for Today's Lecture

1. Explore a **new class of decision functions** (Deep Neural Networks)
2. Consider **variants of this recipe** for training

A Recipe for

2. Choose each of these:
   - Decision function

     $$\hat{y} = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$$

   - Loss function

     $$\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}_i) \in \mathbb{R}$$

4. Train with SGD:
(take small steps opposite the gradient)

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

Idea #1: No pre-training

- **Idea #1: (Just like a shallow network)**
  - Compute the supervised gradient by backpropagation.
  - Take small steps in the direction of the gradient (SGD)

Training

Backpropagation

Backpropagation is just repeated application of the **chain rule** from Calculus 101.

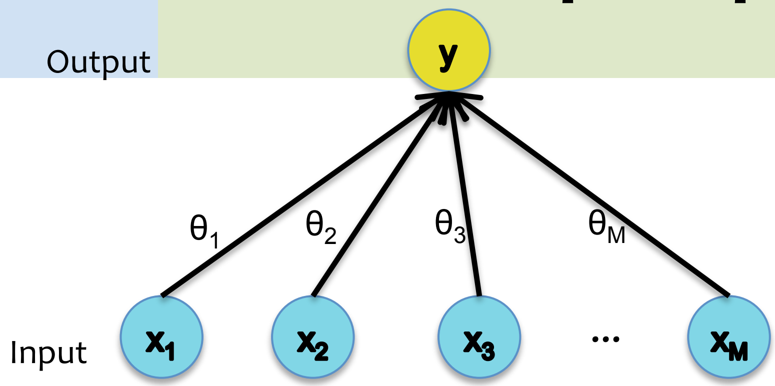
$$\boldsymbol{y} = g(\boldsymbol{u}) \text{ and } \boldsymbol{u} = h(\boldsymbol{x}).$$

Chain Rule:

$$\frac{dy_i}{dx_k} = \sum_{j=1}^{J} \frac{dy_i}{du_j} \frac{du_j}{dx_k}, \quad \forall i, k$$

Training Backpropagation

Case 1:
Logistic
Regression



Forward

$$J = y^* \log q + (1 - y^*) \log(1 - q)$$

$$q = \frac{1}{1 + \exp(-a)}$$

$$a = \sum_{j=0}^{D} \theta_j x_j$$

Backward

$$\frac{dJ}{dq} = \frac{y^*}{q} + \frac{(1 - y^*)}{q - 1}$$

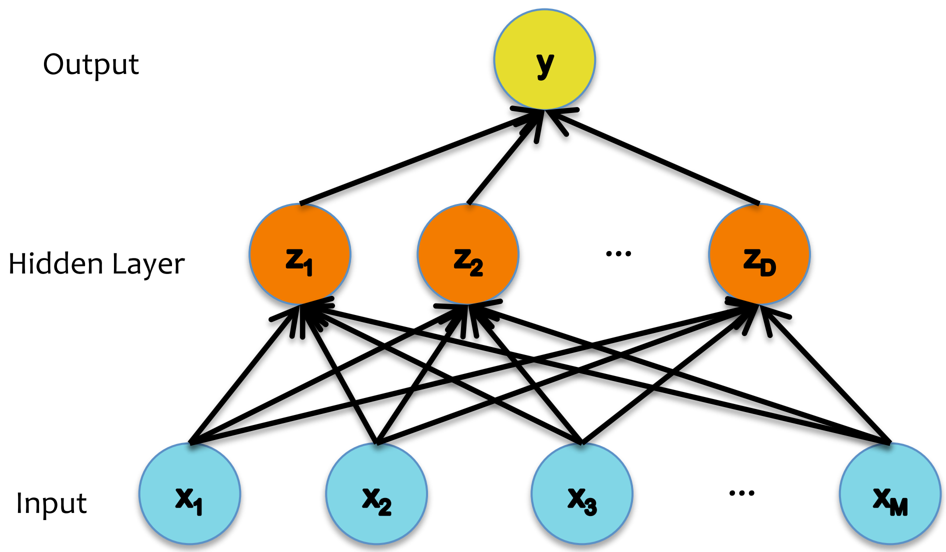
$$\frac{dJ}{da} = \frac{dJ}{dq}\frac{dq}{da}, \quad \frac{dq}{da} = \frac{\exp(a)}{(\exp(a) + 1)^2}$$

$$\frac{dJ}{d\theta_j} = \frac{dJ}{da}\frac{da}{d\theta_j}, \quad \frac{da}{d\theta_j} = x_j$$

$$\frac{dJ}{dx_j} = \frac{dJ}{da}\frac{da}{dx_j}, \quad \frac{da}{dx_j} = \theta_j$$

Training — Backpropagation

What does this picture actually mean?



(F) Loss
$J = \frac{1}{2}(y - y^{(d)})^2$

↑

(E) Output (sigmoid)
$y = \frac{1}{1+\exp(b)}$

↑

(D) Output (linear)
$b = \sum_{j=0}^{D} \beta_j z_j$

↑

(C) Hidden (sigmoid)
$z_j = \frac{1}{1+\exp(a_j)}, \quad \forall j$

↑

(B) Hidden (linear)
$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i, \quad \forall j$

↑

(A) Input
Given $x_i, \quad \forall i$

Training — Backpropagation

Case 2: Neural Network

Forward

$$J = y^* \log q + (1 - y^*) \log(1 - q)$$

$$q = \frac{1}{1 + \exp(-b)}$$

$$b = \sum_{j=0}^{D} \beta_j z_j$$

$$z_j = \frac{1}{1 + \exp(-a_j)}$$

$$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i$$

Backward

$$\frac{dJ}{dq} = \frac{y^*}{q} + \frac{(1 - y^*)}{q - 1}$$

$$\frac{dJ}{db} = \frac{dJ}{dy}\frac{dy}{db}, \quad \frac{dy}{db} = \frac{\exp(b)}{(\exp(b) + 1)^2}$$

$$\frac{dJ}{d\beta_j} = \frac{dJ}{db}\frac{db}{d\beta_j}, \quad \frac{db}{d\beta_j} = z_j$$

$$\frac{dJ}{dz_j} = \frac{dJ}{db}\frac{db}{dz_j}, \quad \frac{db}{dz_j} = \beta_j$$

$$\frac{dJ}{da_j} = \frac{dJ}{dz_j}\frac{dz_j}{da_j}, \quad \frac{dz_j}{da_j} = \frac{\exp(a_j)}{(\exp(a_j) + 1)^2}$$

$$\frac{dJ}{d\alpha_{ji}} = \frac{dJ}{da_j}\frac{da_j}{d\alpha_{ji}}, \quad \frac{da_j}{d\alpha_{ji}} = x_i$$

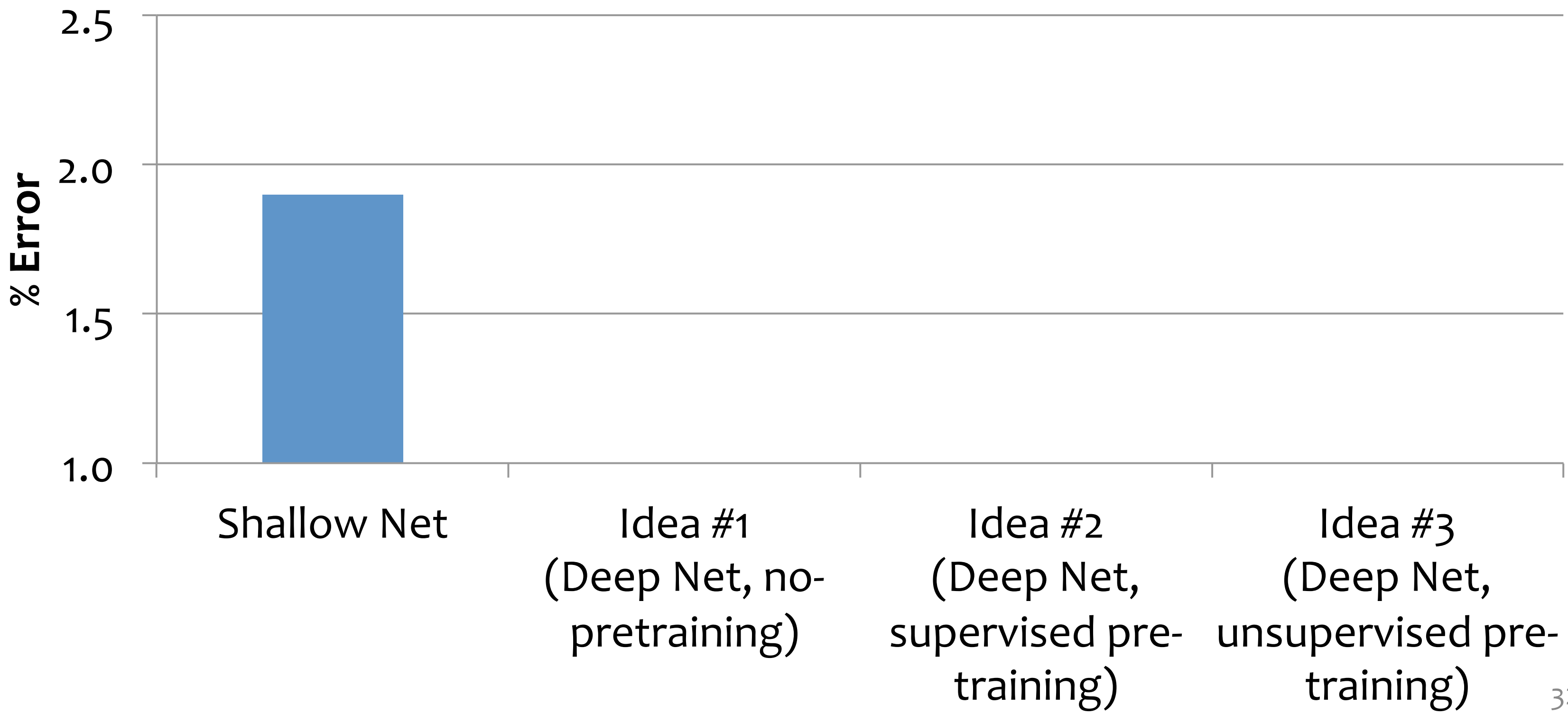
$$\frac{dJ}{dx_i} = \frac{dJ}{da_j}\frac{da_j}{dx_i}, \quad \frac{da_j}{dx_i} = \sum_{j=0}^{D} \alpha_{ji}$$

Idea #1: No pre-training

- **Idea #1: (Just like a shallow network)**
  - Compute the supervised gradient by backpropagation.
  - Take small steps in the direction of the gradient (SGD)

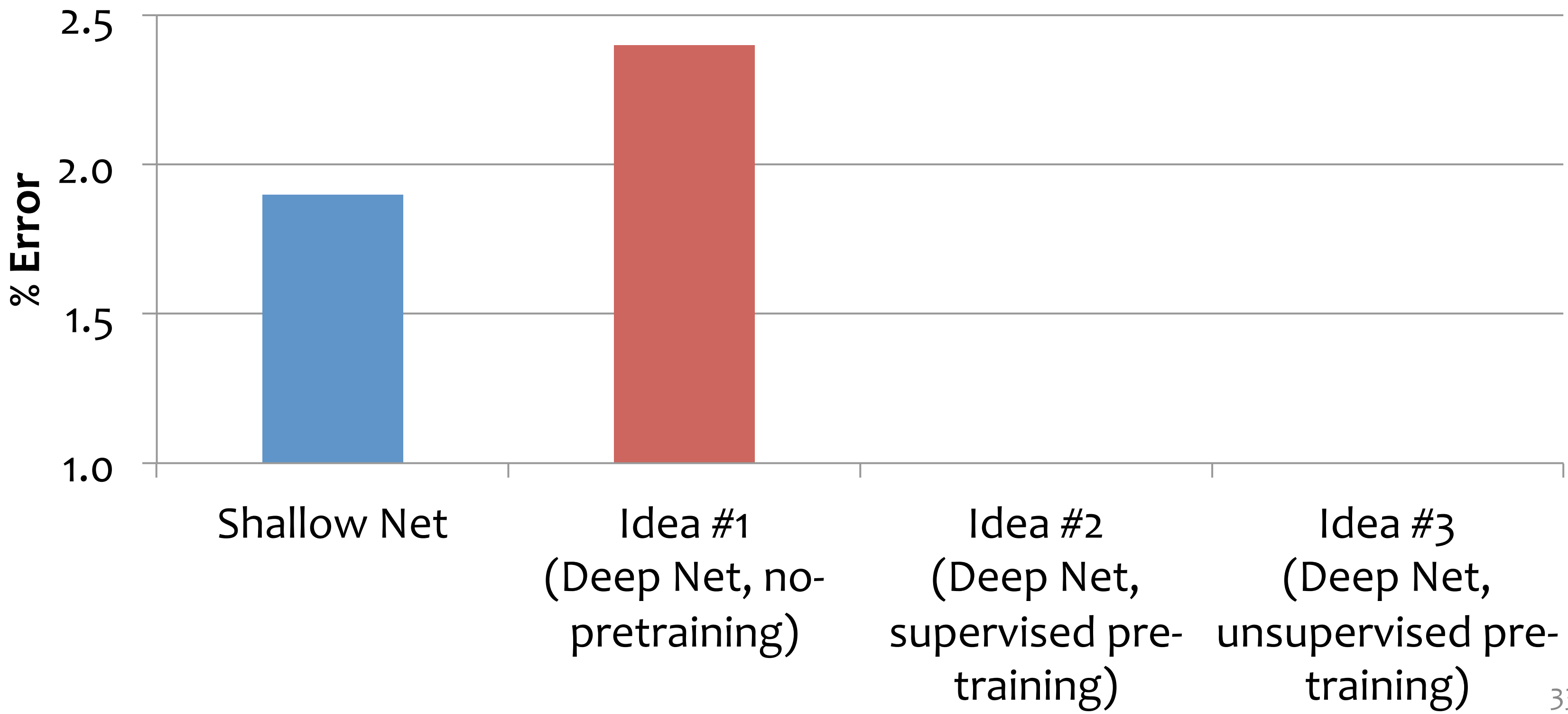
Comparison on MNIST

- Results from Bengio et al. (2006) on MNIST digit classification task
- Percent error (lower is better)



Comparison on MNIST

- Results from Bengio et al. (2006) on MNIST digit classification task
- Percent error (lower is better)



Idea #1: No pre-training

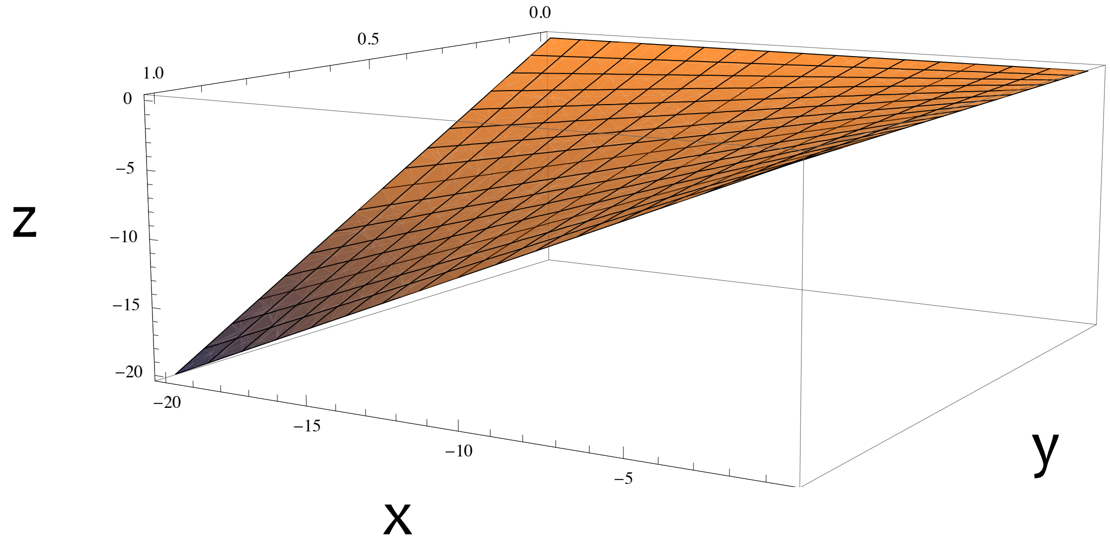
- **Idea #1: (Just like a shallow network)**
  - Compute the supervised gradient by backpropagation.
  - Take small steps in the direction of the gradient (SGD)

- What goes wrong?
  A. Gets stuck in local optima
     - Nonconvex objective
     - Usually start at a random (bad) point in parameter space
  B. Gradient is progressively getting more dilute
     - "Vanishing gradients"

Training

Problem A: *Nonconvexity*

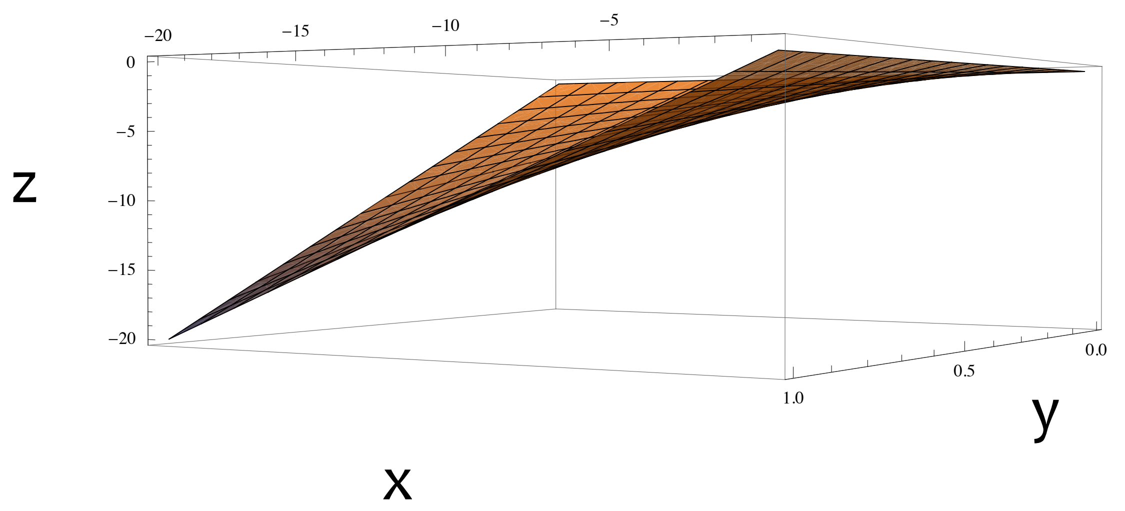
- Where does the nonconvexity come from?
- Even a simple quadratic z = xy objective is nonconvex:



Training

Problem A: *Nonconvexity*

- Where does the nonconvexity come from?
- Even a simple quadratic z = xy objective is nonconvex:

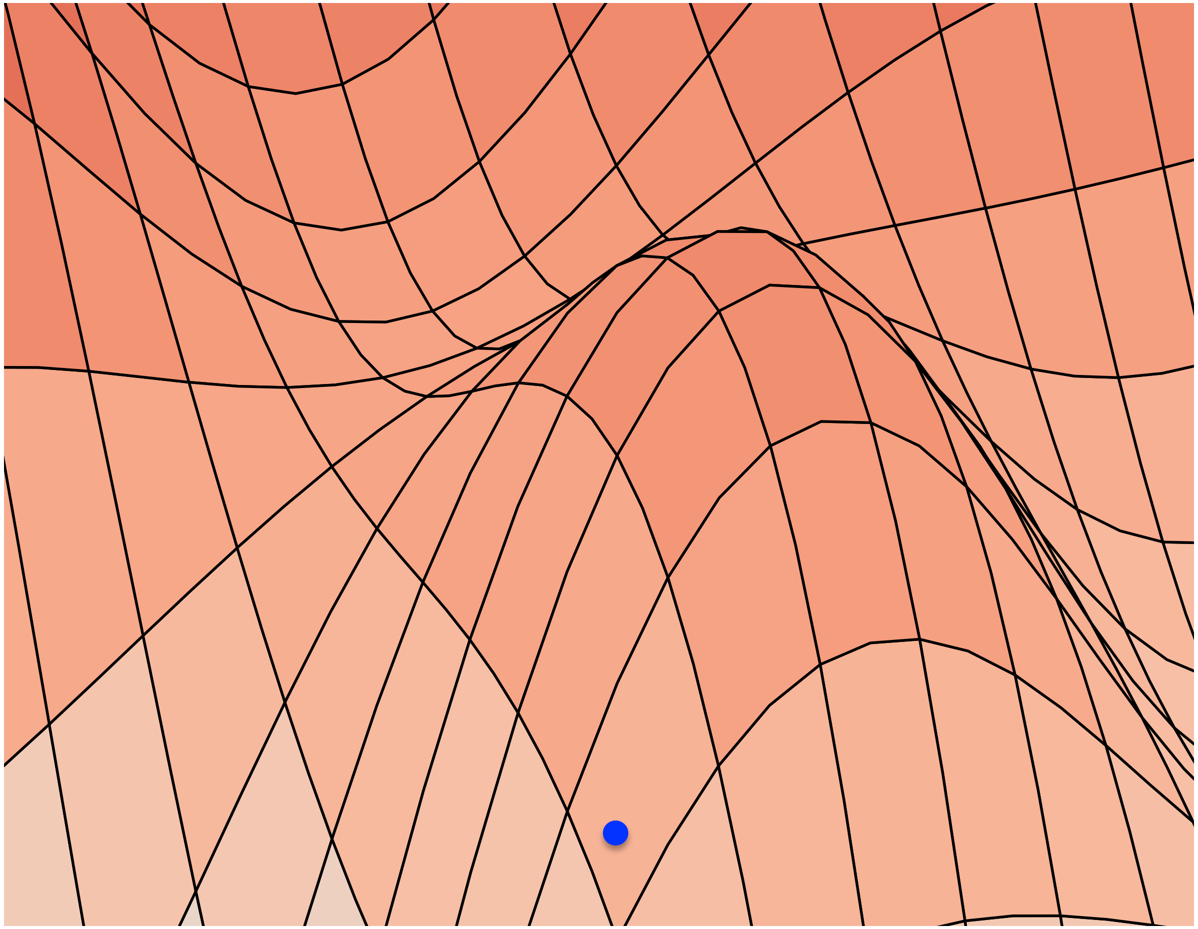


Training

Problem A: *Nonconvexity*

Stochastic Gradient Descent…

… climbs to the top of the nearest hill…

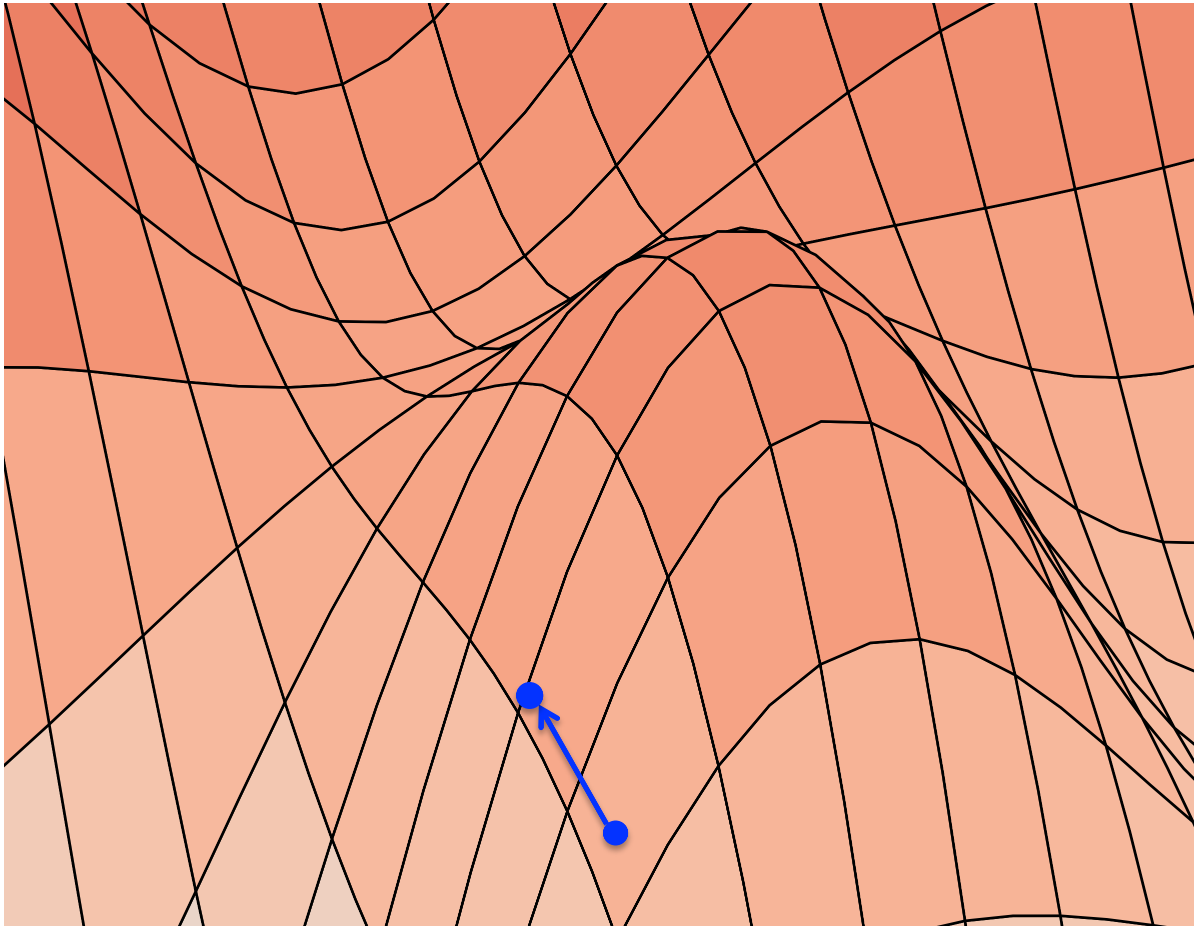


Training

Problem A: *Nonconvexity*

Stochastic Gradient Descent…

…climbs to the top of the nearest hill…

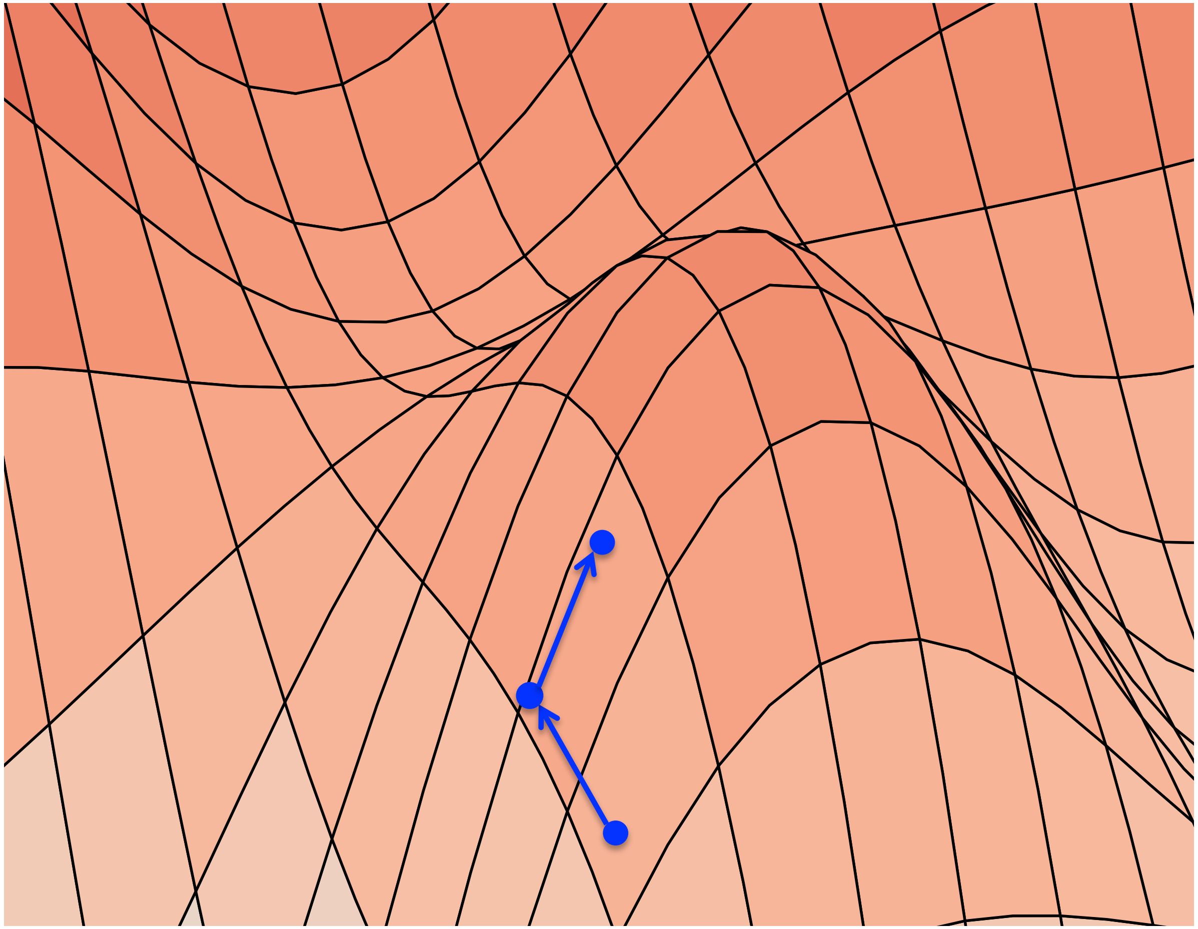


Training

Problem A: *Nonconvexity*

Stochastic Gradient Descent…

…climbs to the top of the nearest hill…

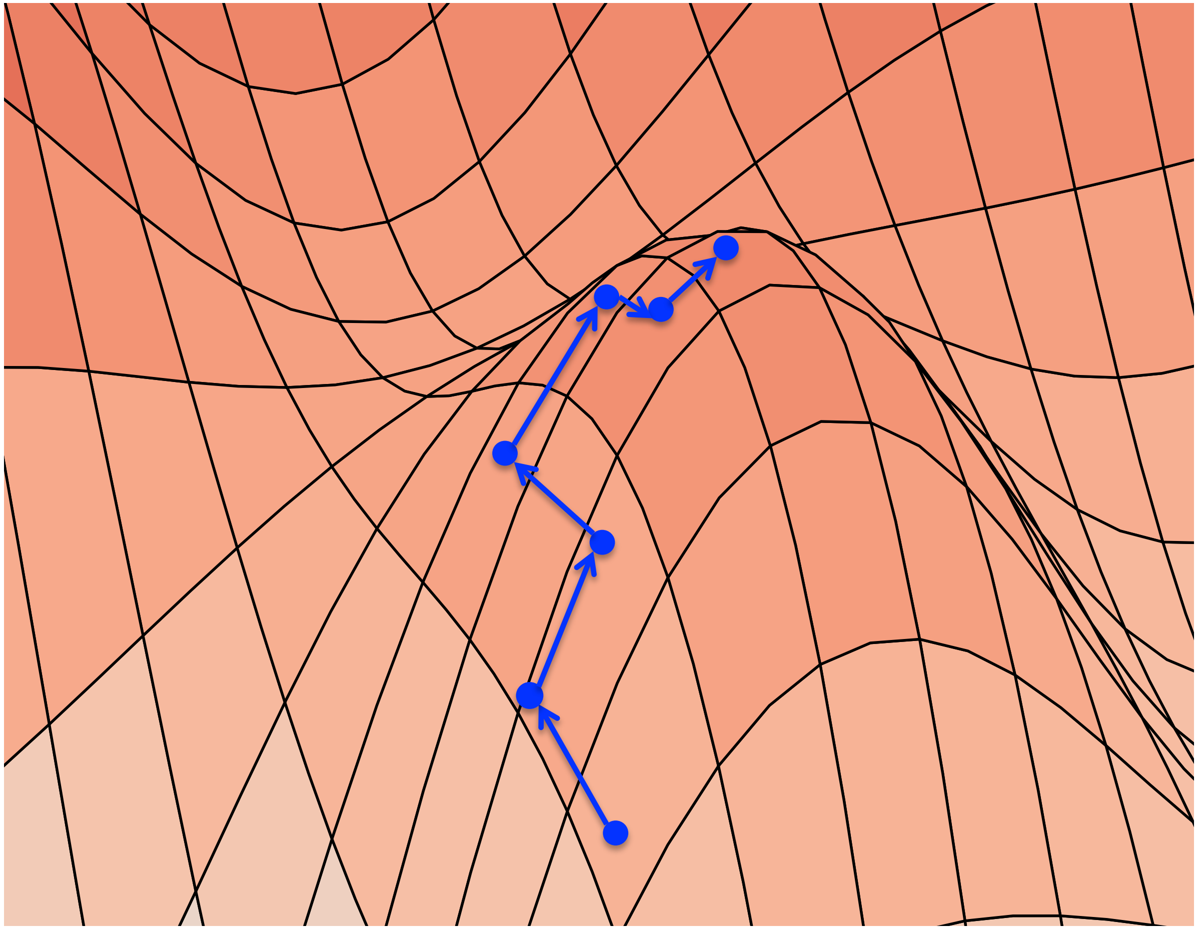


Training

Problem A: *Nonconvexity*

Stochastic Gradient Descent…

…climbs to the top of the nearest hill…



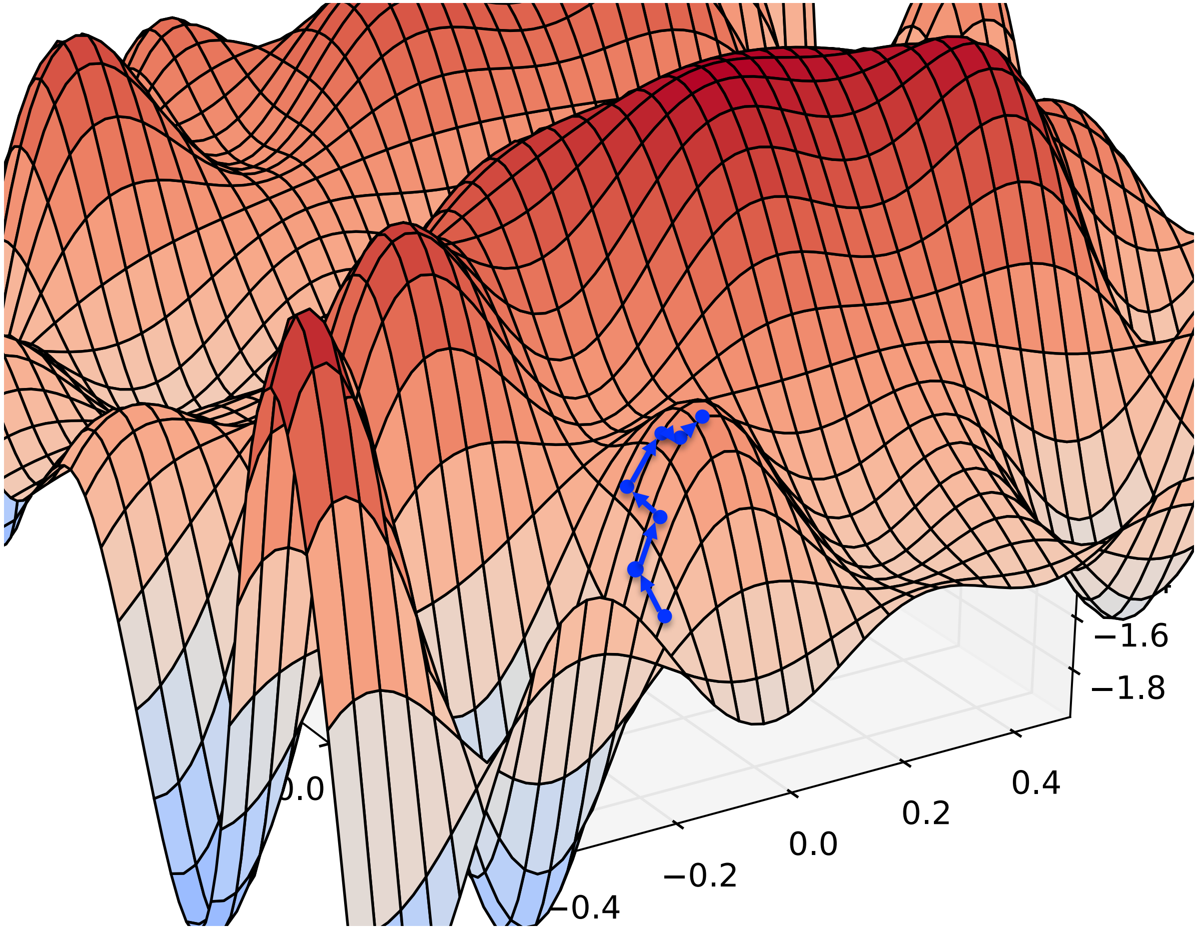
Training

Problem A: *Nonconvexity*

Stochastic Gradient Descent…

… climbs to the top of the nearest hill…

… which might not lead to the top of the mountain

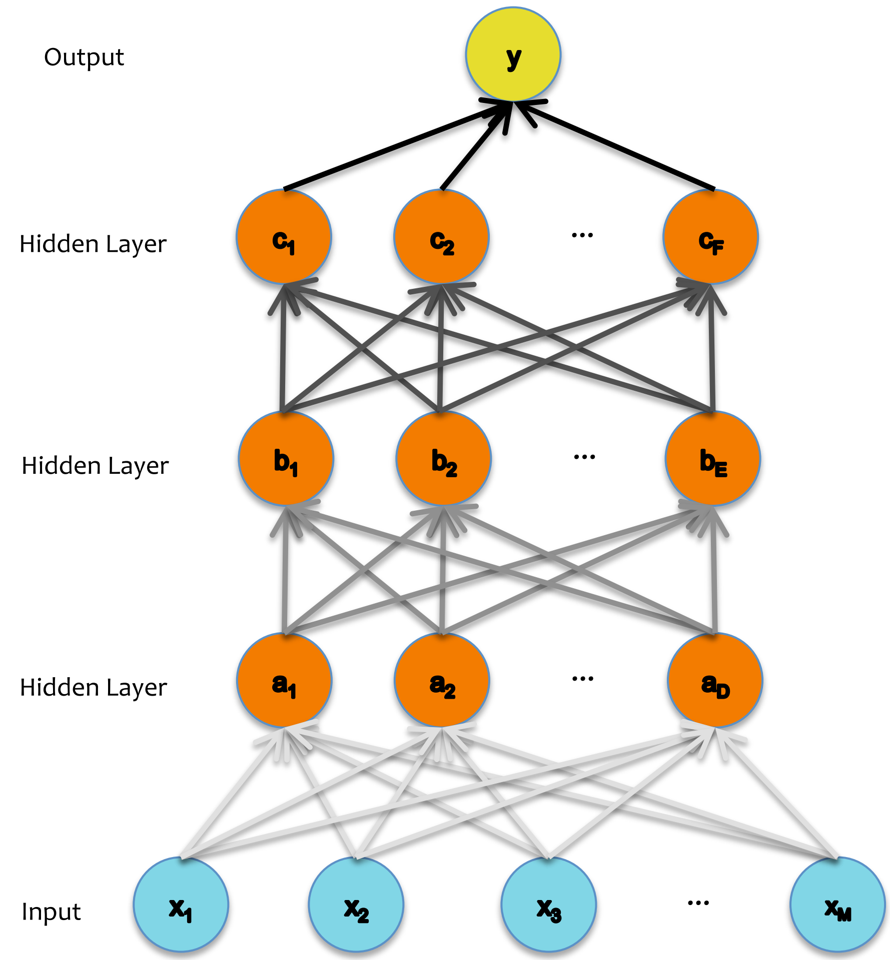


Training

Problem B: *Vanishing Gradients*

The gradient for an edge at the base of the network depends on the gradients of many edges above it

The chain rule multiplies many of these partial derivatives together

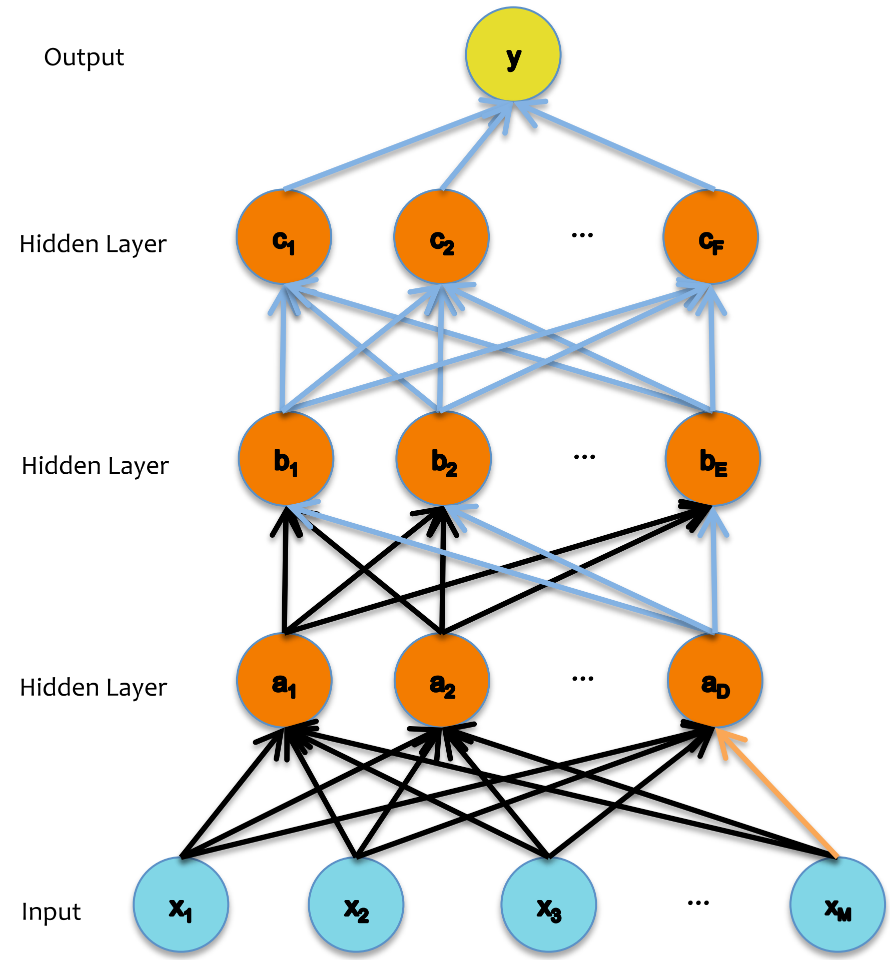


Training

Problem B: *Vanishing Gradients*

The gradient for an edge at the base of the network depends on the gradients of many edges above it

The chain rule multiplies many of these partial derivatives together

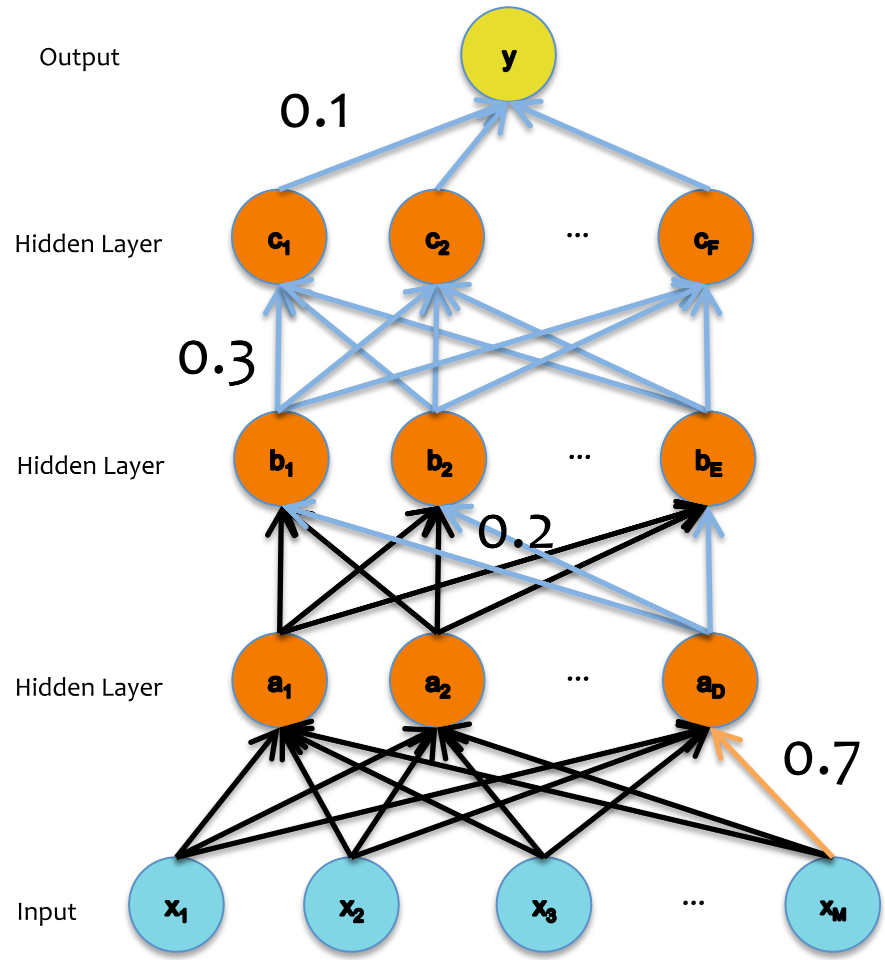


Training

Problem B: *Vanishing Gradients*

The gradient for an edge at the base of the network depends on the gradients of many edges above it

The chain rule multiplies many of these partial derivatives together



Training

Idea #1: No pre-training

- **Idea #1: (Just like a shallow network)**
  - Compute the supervised gradient by backpropagation.
  - Take small steps in the direction of the gradient (SGD)

- What goes wrong?

  A. Gets stuck in local optima
    - Nonconvex objective
    - Usually start at a random (bad) point in parameter space

  B. Gradient is progressively getting more dilute
    - "Vanishing gradients"

Idea #2: Supervised Pre-training

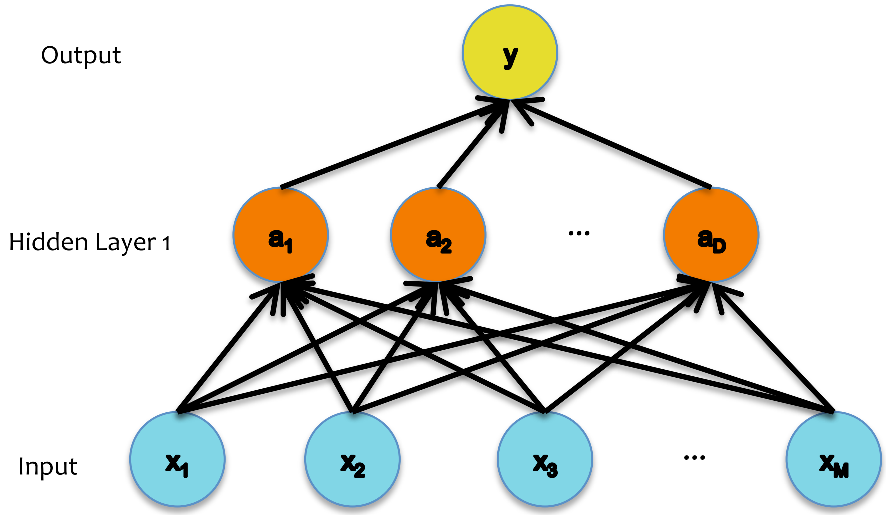
- Idea #2: (Two Steps)
  - **Train each level** of the model in a **greedy** way
  - Then use our **original idea**

1. Supervised Pre-training
   - Use **labeled** data
   - Work bottom-up
     - Train hidden layer 1. Then fix its parameters.
     - Train hidden layer 2. Then fix its parameters.
     - …
     - Train hidden layer n. Then fix its parameters.
2. Supervised Fine-tuning
   - Use **labeled** data to train following "Idea #1"
   - Refine the features by backpropagation so that they become tuned to the end-task

Training

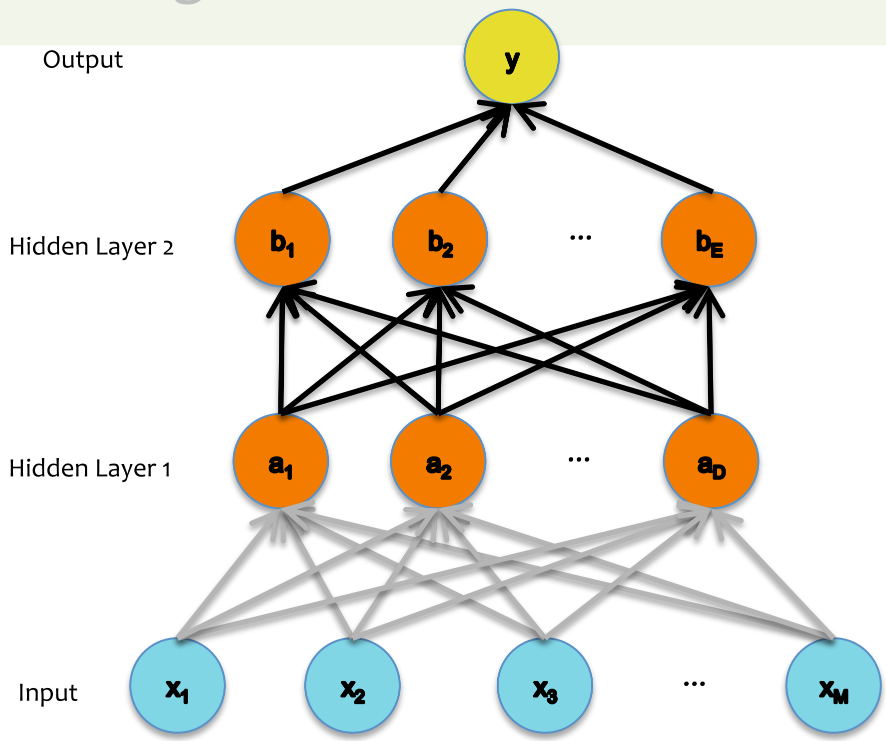
Idea #2: Supervised Pre-training

- **Idea #2: (Two Steps)**
  - **Train each level** of the model in a **greedy** way
  - Then use our **original idea**



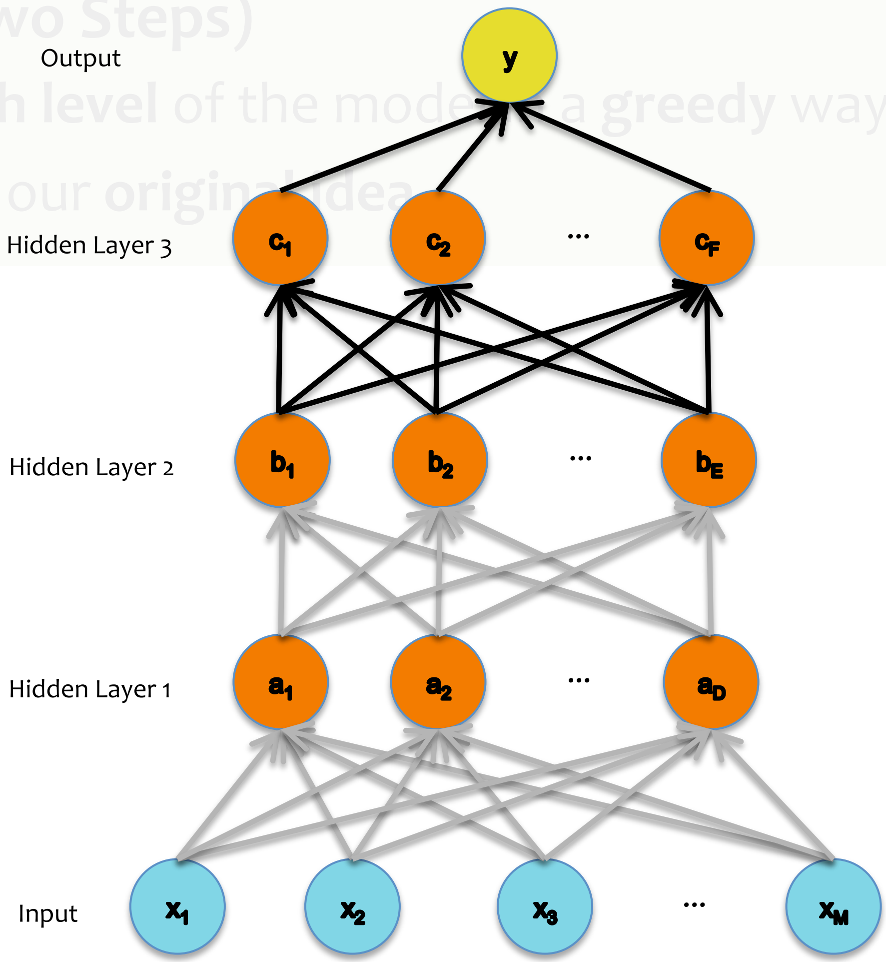
Idea #2: Supervised Pre-training

- Idea #2: (Two Steps)
  - **Train each level** of the model in a **greedy** way
  - Then use our **original idea**



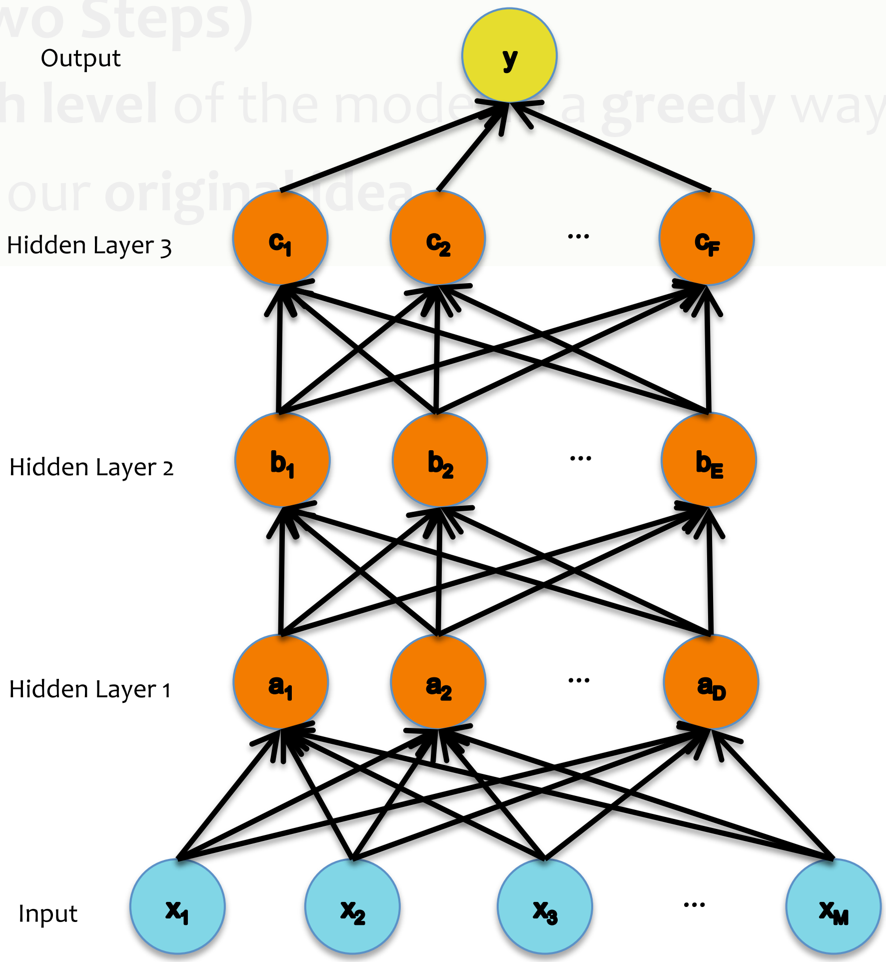
Idea #2: Supervised Pre-training

- Idea #2: (Two Steps)
  - **Train each level** of the model in a **greedy** way
  - Then use our **original idea**



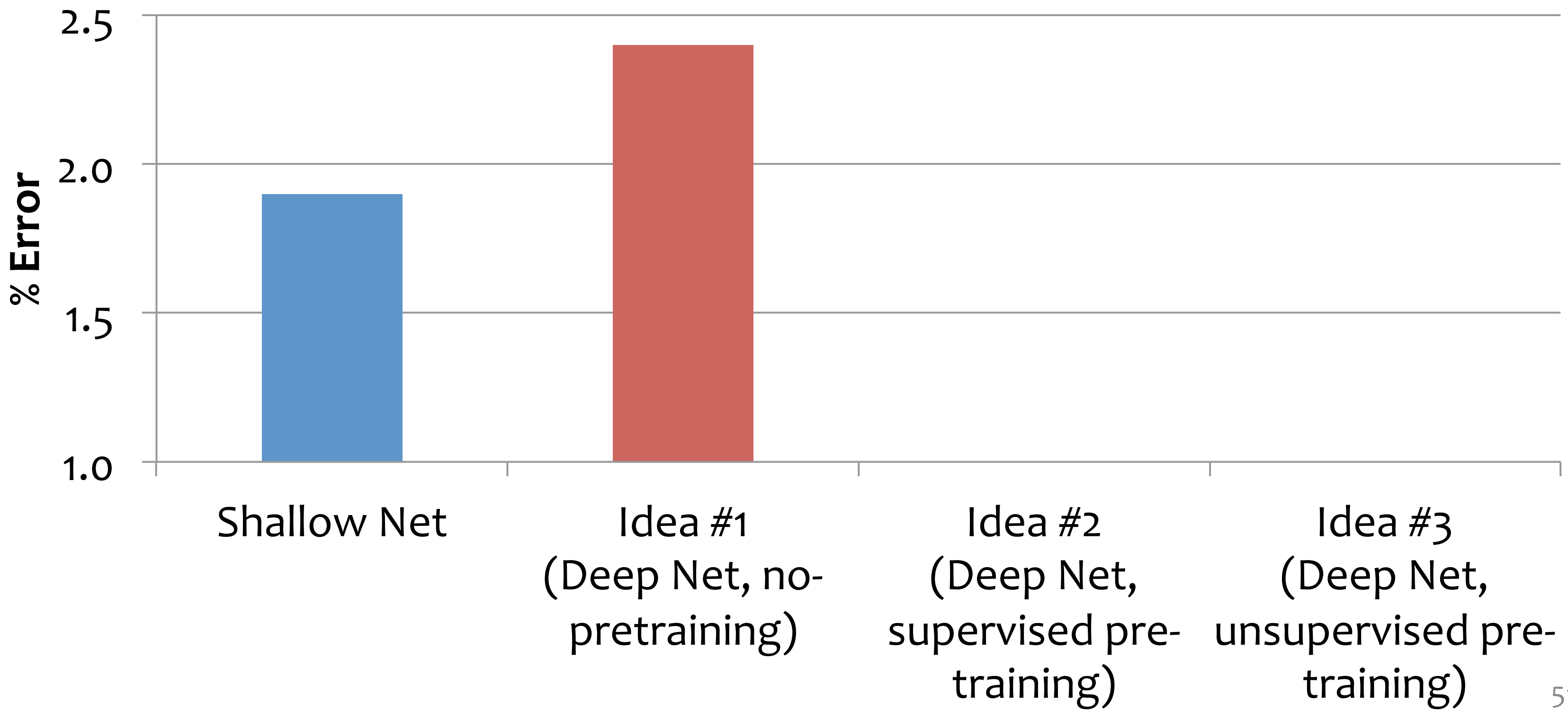
Idea #2: Supervised Pre-training

- Idea #2: (Two Steps)
  - **Train each level** of the model in a **greedy** way
  - Then use our **original idea**



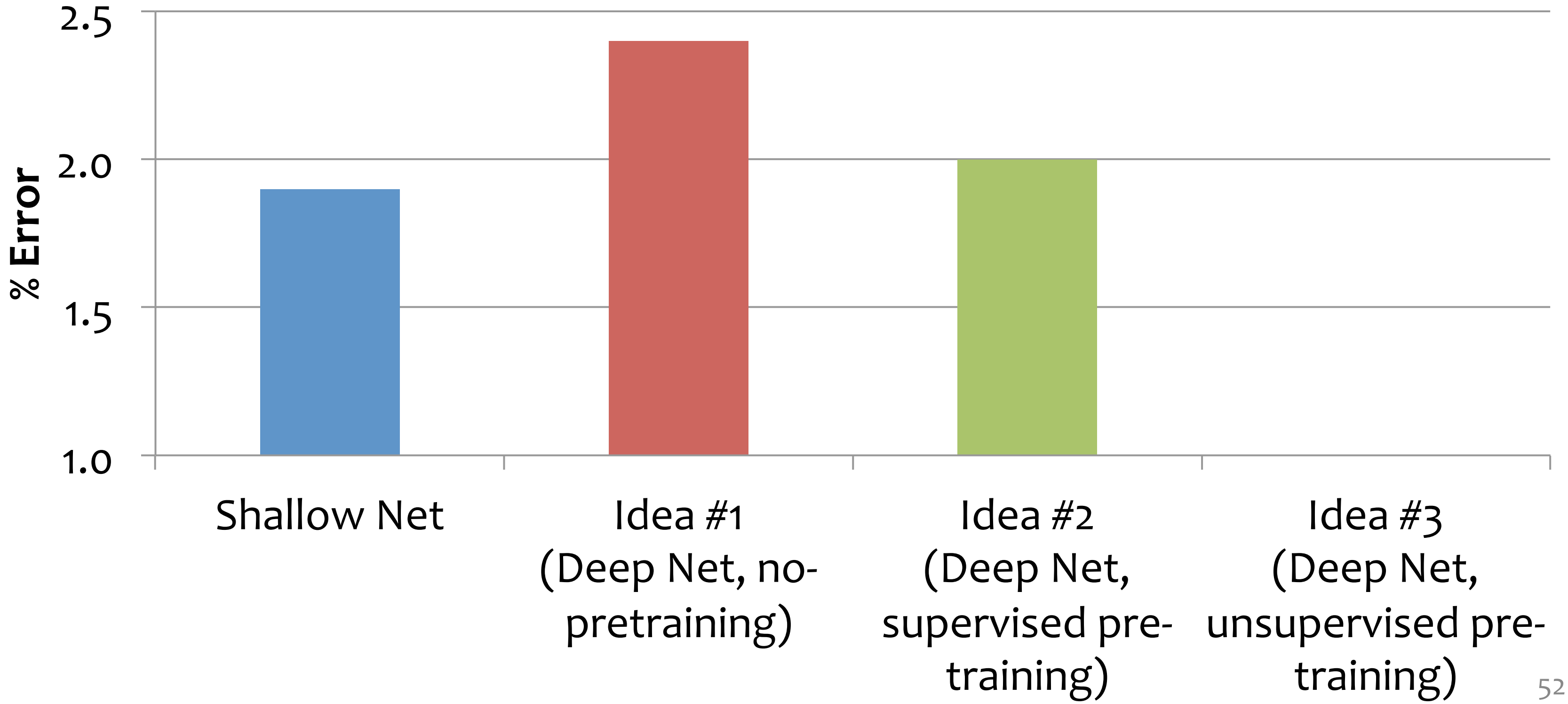
Comparison on MNIST

- Results from Bengio et al. (2006) on MNIST digit classification task
- Percent error (lower is better)



Comparison on MNIST

- Results from Bengio et al. (2006) on MNIST digit classification task
- Percent error (lower is better)



Training

Idea #3: Unsupervised Pre-training

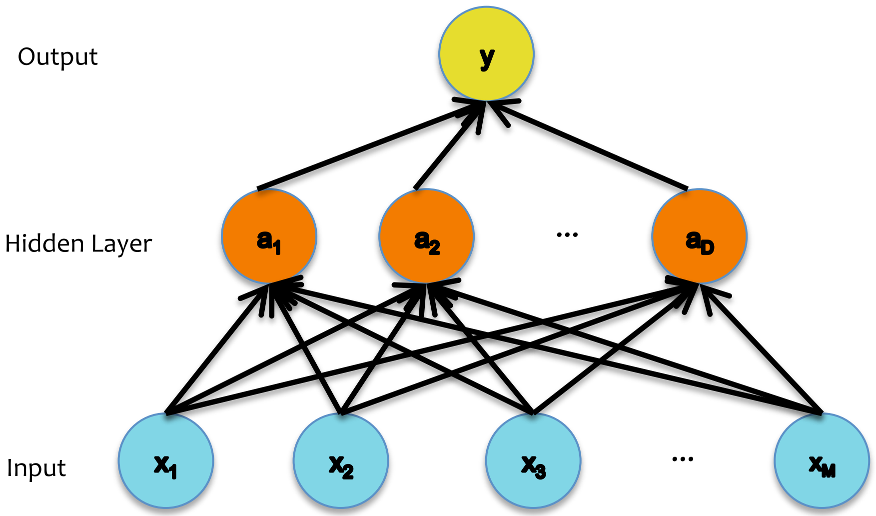
- **Idea #3: (Two Steps)**
  - Use our original idea, but **pick a better starting point**
  - **Train each level** of the model in a **greedy** way

1. Unsupervised Pre-training
   – Use **unlabeled** data
   – Work bottom-up
     - Train hidden layer 1. Then fix its parameters.
     - Train hidden layer 2. Then fix its parameters.
     - …
     - Train hidden layer n. Then fix its parameters.
2. Supervised Fine-tuning
   – Use **labeled** data to train following "Idea #1"
   – Refine the features by backpropagation so that they become tuned to the end-task

The solution: *Unsupervised pre-training*

Unsupervised pre-training of the first layer:

- What should it predict?
- What else do we observe?
- **The input!**

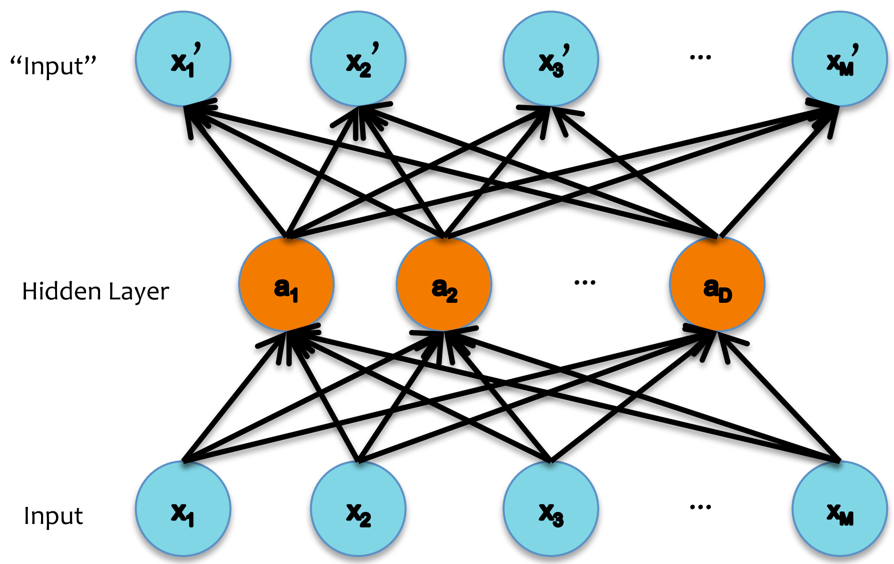


The solution: *Unsupervised pre-training*

Unsupervised pre-training of the first layer:

- What should it predict?
- What else do we observe?
- **The input!**

This topology defines an Auto-encoder.



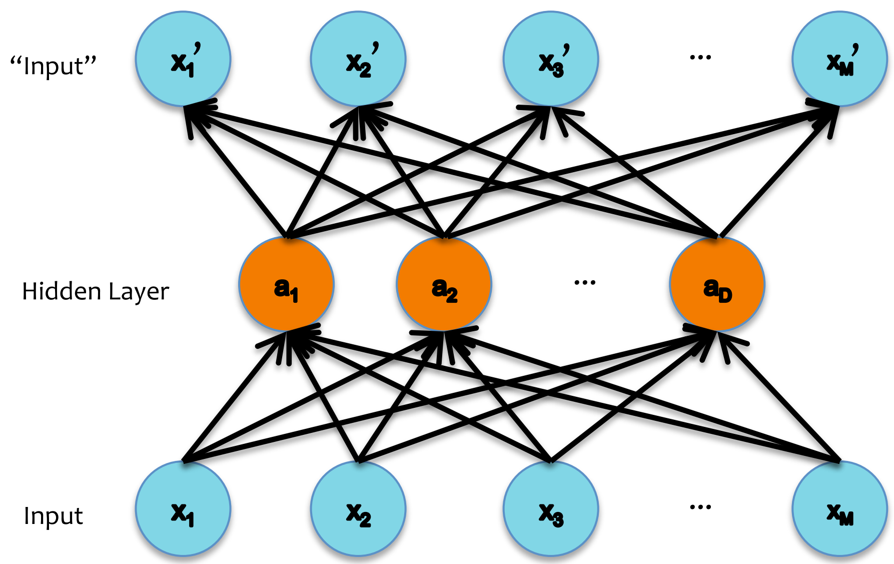
Auto-Encoders

Key idea: Encourage z to give small reconstruction error:

- x’ is the *reconstruction* of x
- Loss = || x – DECODER(ENCODER(x)) ||²
- Train with the same backpropagation algorithm for 2-layer Neural Networks with x_m as both input and output.

DECODER: x’ = h(W’z)

ENCODER: z = h(Wx)

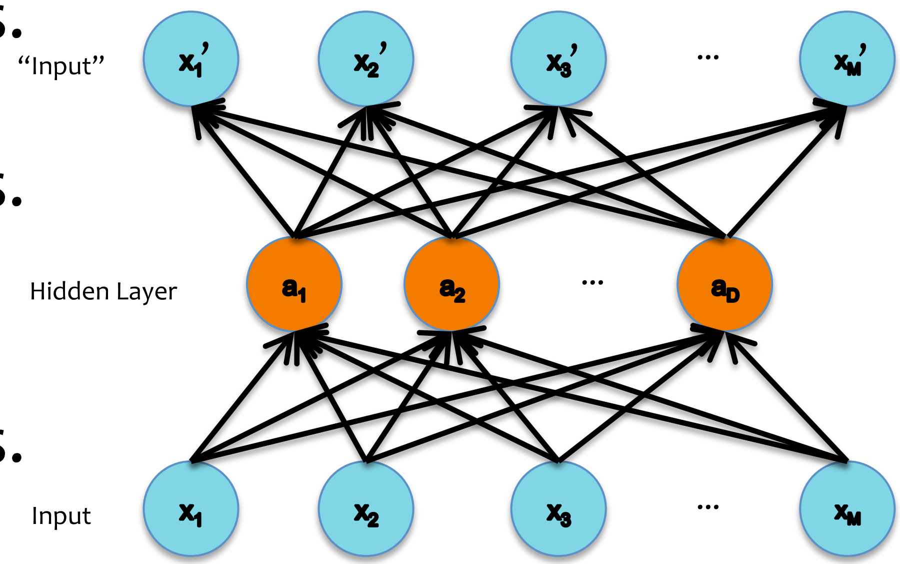


Slide adapted from Raman Arora

The solution: *Unsupervised pre-training*

Unsupervised pre-training

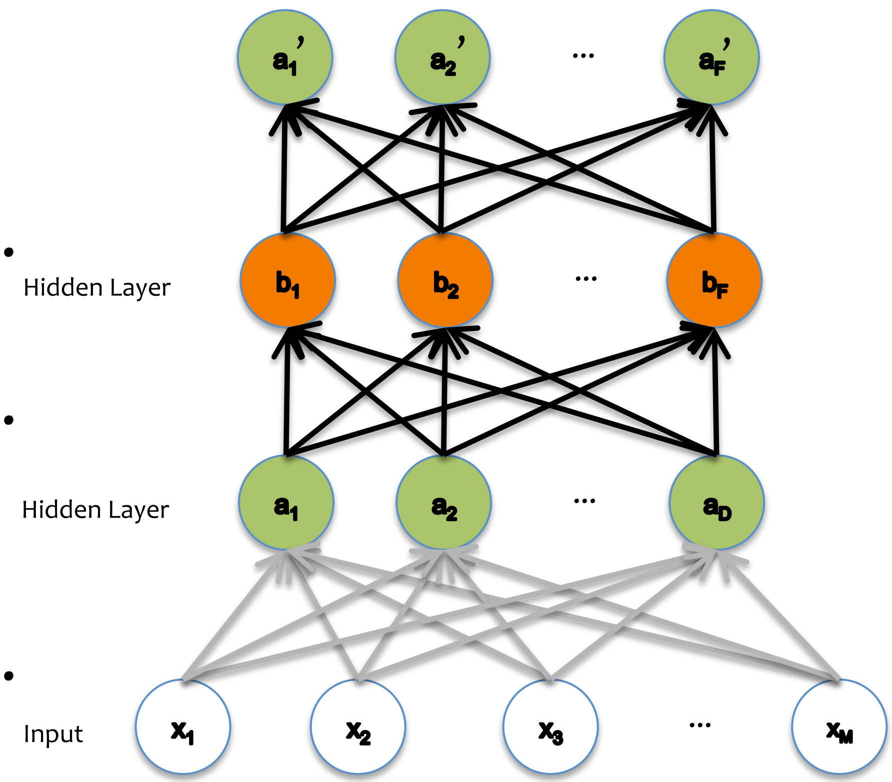
- Work bottom-up
  - Train hidden layer 1. Then fix its parameters.
  - Train hidden layer 2. Then fix its parameters.
  - …
  - Train hidden layer n. Then fix its parameters.



The solution: *Unsupervised pre-training*

Unsupervised pre-training

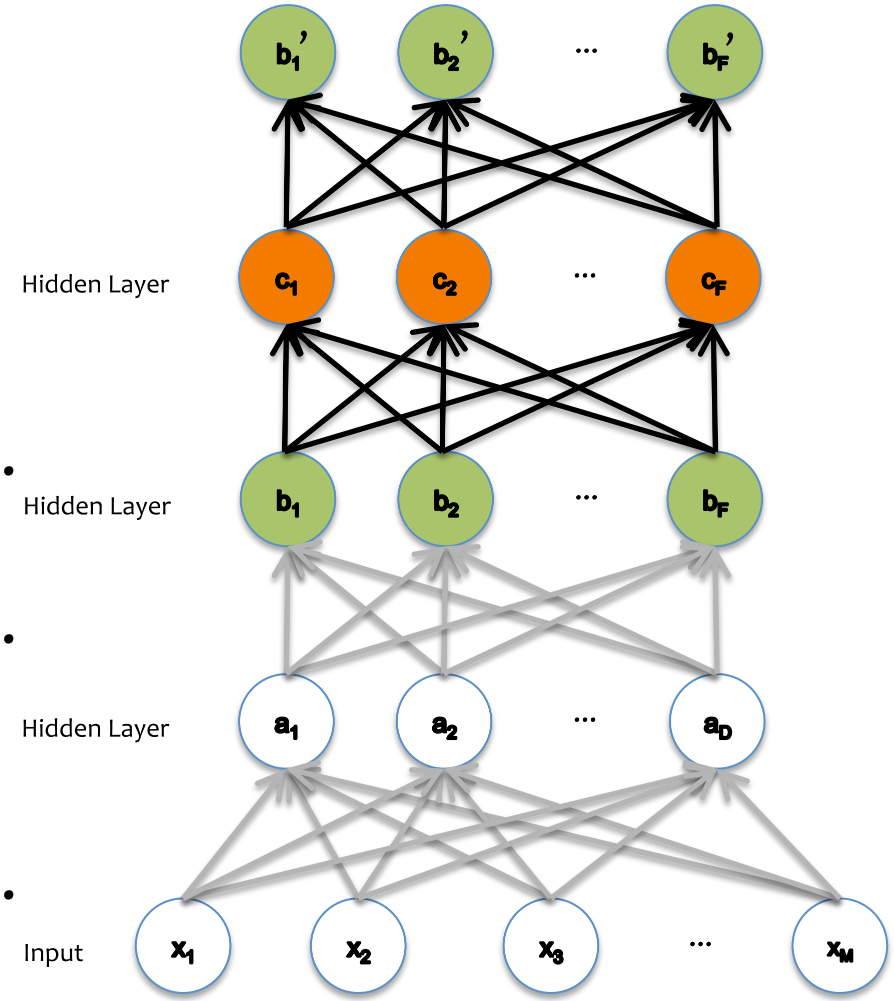
- Work bottom-up
  - Train hidden layer 1. Then fix its parameters.
  - Train hidden layer 2. Then fix its parameters.
  - …
  - Train hidden layer n. Then fix its parameters.



The solution: *Unsupervised pre-training*

Unsupervised pre-training

- Work bottom-up
  - Train hidden layer 1. Then fix its parameters.
  - Train hidden layer 2. Then fix its parameters.
  - …
  - Train hidden layer n. Then fix its parameters.



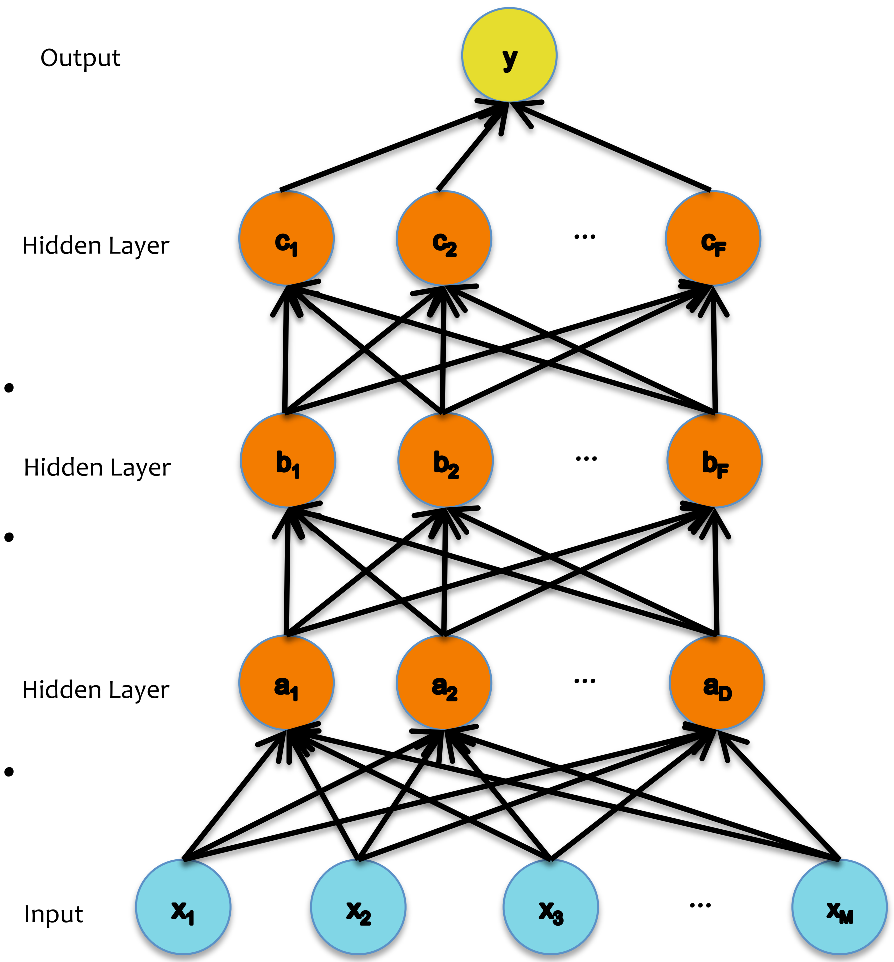
The solution: *Unsupervised pre-training*

Unsupervised pre-training

- Work bottom-up
  - Train hidden layer 1. Then fix its parameters.
  - Train hidden layer 2. Then fix its parameters.
  - …
  - Train hidden layer n. Then fix its parameters.

Supervised fine-tuning

Backprop and update all parameters



Deep Network Training

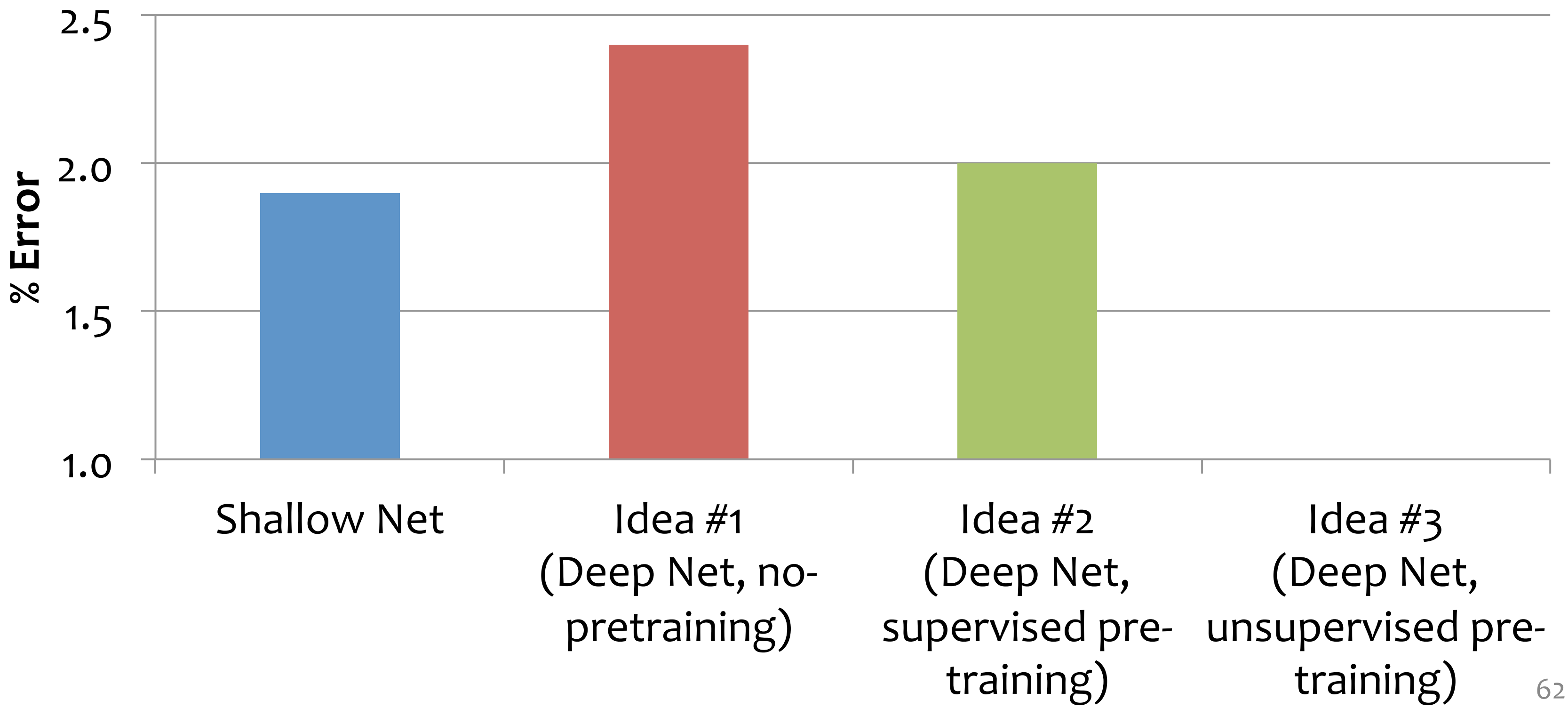
- **Idea #1:**
  1. Supervised fine-tuning only

- **Idea #2:**
  1. Supervised layer-wise pre-training
  2. Supervised fine-tuning

- **Idea #3:**
  1. Unsupervised layer-wise pre-training
  2. Supervised fine-tuning

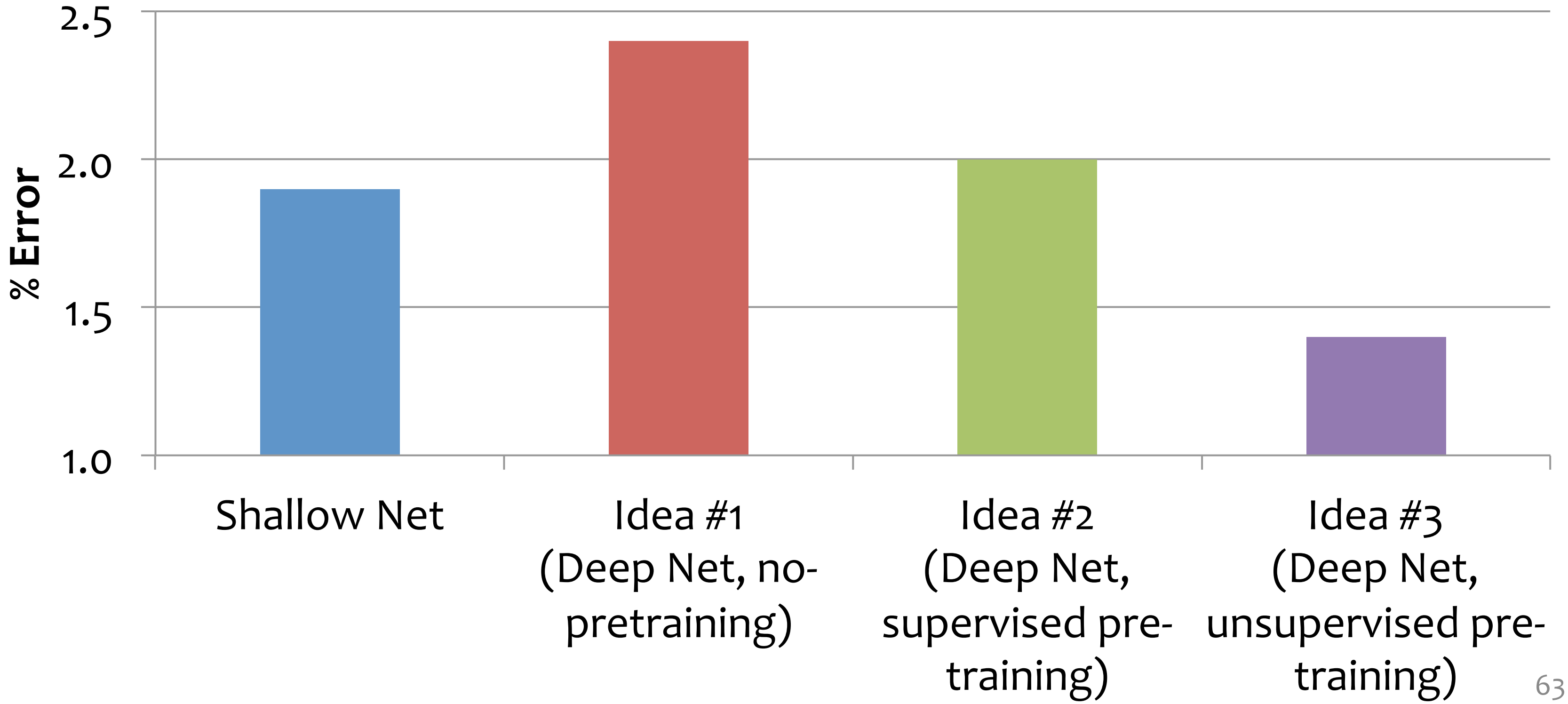
Comparison on MNIST

- Results from Bengio et al. (2006) on MNIST digit classification task
- Percent error (lower is better)



Comparison on MNIST

- Results from Bengio et al. (2006) on MNIST digit classification task
- Percent error (lower is better)



Is layer-wise pre-training always necessary?

In 2010, a record on a hand-writing recognition task was set by standard supervised backpropagation (our Idea #1).

How? A very fast implementation on GPUs.

See Ciresen et al. (2010)

Deep Learning

- Goal: learn features at different levels of abstraction
- Training can be tricky due to…
  - Nonconvexity
  - Vanishing gradients
- Unsupervised layer-wise pre-training can help with both!

Outline

- **Motivation**
- **Deep Neural Networks (DNNs)**
  - Background: Decision functions
  - Background: Neural Networks
  - Three ideas for training a DNN
  - Experiments: MNIST digit classification
- **Deep Belief Networks (DBNs)**
  - Sigmoid Belief Network
  - Contrastive Divergence learning
  - Restricted Boltzman Machines (RBMs)
  - RBMs as infinitely deep Sigmoid Belief Nets
  - Learning DBNs
- **Deep Boltzman Machines (DBMs)**
  - Boltzman Machines
  - Learning Boltzman Machines
  - Learning DBMs

Question:

How does this relate to Graphical Models?

The first "Deep Learning" papers in 2006 were innovations in training a particular flavor of Belief Network.

Those models happen to also be neural nets.

DBNs

MNIST Digit **Generation**

- This section: Suppose you want to build a **generative** model capable of explaining handwritten digits
- Goal:
  - To have a model p(x) **from which we can sample digits** that look realistic
  - Learn **unsupervised** hidden representation of an image

Figure from (Hinton et al., 2006)

Sigmoid Belief Networks

- Directed graphical model of binary variables in fully connected layers
- Only bottom layer is observed
- Specific parameterization of the conditional probabilities:

$$p(x_i | \text{parents}(x_i)) = \frac{1}{1 + \exp(-\sum_j w_{ij} x_j)}$$

Note: this is a GM diagram not a NN!

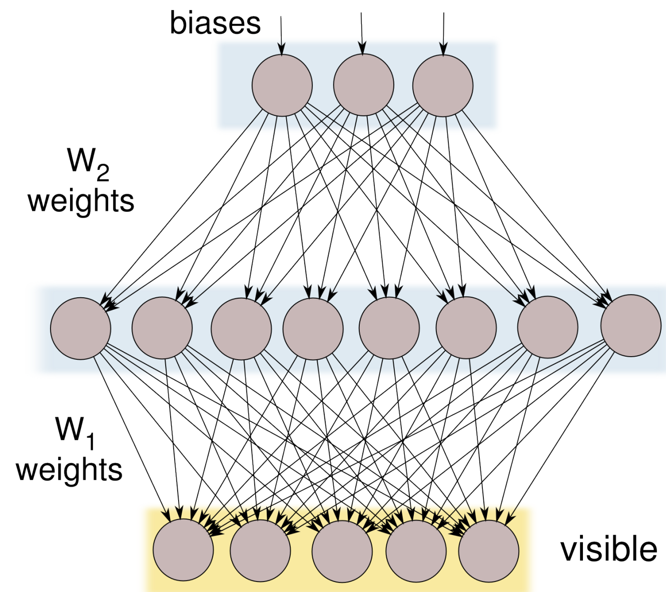


Figure from Marcus Frean, MLSS Tutorial 2010

Contrastive Divergence Training

Contrastive Divergence is a general tool for learning a generative distribution, where the derivative of the log partition function is intractable to compute.

$$
\begin{aligned}
\log L &= \log P(\mathcal{D}) \\
&= \sum_{\mathbf{v} \in \mathcal{D}} \log P(\mathbf{v}) \\
&= \sum_{\mathbf{v} \in \mathcal{D}} \log \left(P^{\star}(\mathbf{v}) / Z \right) \\
&= \sum_{\mathbf{v} \in \mathcal{D}} \left(\log P^{\star}(\mathbf{v}) \quad - \quad \log Z \right) \\
&\propto \quad \frac{1}{N} \sum_{\mathbf{v} \in \mathcal{D}} \log P^{\star}(\mathbf{v}) \qquad - \quad \log Z
\end{aligned}
$$

Slide from Marcus Frean, MLSS Tutorial 2010

Contrastive Divergence Training

$$\frac{\partial}{\partial w} \log L \quad \propto$$

$$\underbrace{\frac{1}{N} \sum_{\mathbf{v} \in \mathcal{D}}}_{\text{data}} \underbrace{\sum_{\mathbf{h}} P(\mathbf{h} \mid \mathbf{v})}_{\text{av. over posterior}} \frac{\partial}{\partial w} \log P^\star(\mathbf{x}) \quad - \quad \underbrace{\sum_{\mathbf{v},\mathbf{h}} P(\mathbf{v}, \mathbf{h})}_{\text{av. over joint}} \frac{\partial}{\partial w} \log P^\star(\mathbf{x})$$

Contrastive Divergence estimates the second term with a Monte Carlo estimate from 1-step of a Gibbs sampler!

Both terms involve averaging over $\frac{\partial}{\partial w} \log P^\star(\mathbf{x})$.

Another way to write it:

$$\left\langle \frac{\partial}{\partial w} \log P^\star(\mathbf{x}) \right\rangle_{\mathbf{v} \in \mathcal{D},\ \mathbf{h} \sim P(\mathbf{h}|\mathbf{v})} \quad - \quad \left\langle \frac{\partial}{\partial w} \log P^\star(\mathbf{x}) \right\rangle_{\mathbf{x} \sim P(\mathbf{x})}$$

clamped / wake phase	unclamped / sleep / free phase
↑↑↑ conditioned hypotheses	↓↓↓ random fantasies

Contrastive Divergence Training

For a belief net the joint is automatically normalised: Z is a constant 1

- 2nd term is zero!
- for the weight w_{ij} from j into i, the gradient $\frac{\partial \log L}{\partial w_{ij}} = (x_i - p_i)x_j$
- stochastic gradient ascent:

$$\Delta w_{ij} \quad \propto \quad \underbrace{(x_i - p_i)x_j}_{\text{the "delta rule"}}$$

So this is a stochastic version of the EM algorithm, that you may have heard of. We iterate the following two steps:

E step: get samples from the posterior

M step: apply the learning rule that makes them more likely

Slide from Marcus Frean, MLSS Tutorial 2010

Sigmoid Belief Networks

- In practice, applying CD to a Deep Sigmoid Belief Nets fails
- Sampling from the posterior of many (deep) hidden layers doesn't approach the equilibrium distribution quickly enough

Note: this is a GM diagram not a NN!

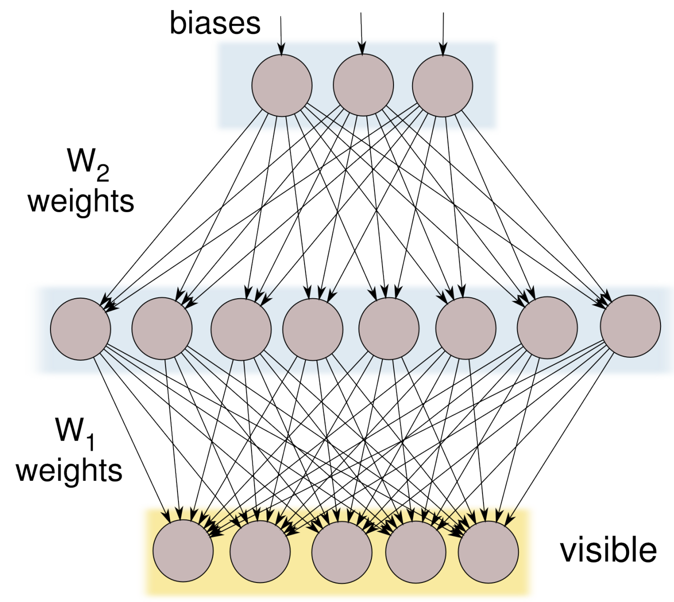


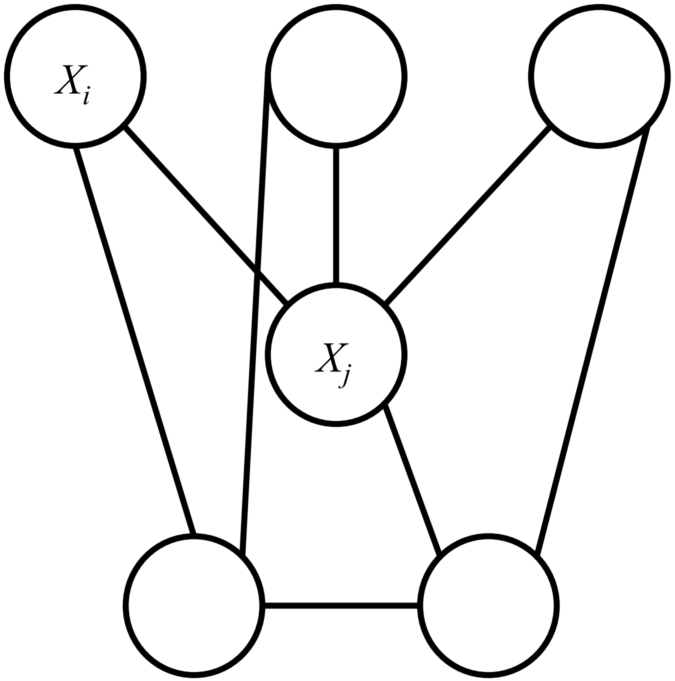
Figure from Marcus Frean, MLSS Tutorial 2010

Boltzman Machines

- Undirected graphical model of binary variables with pairwise potentials
- Parameterization of the potentials:

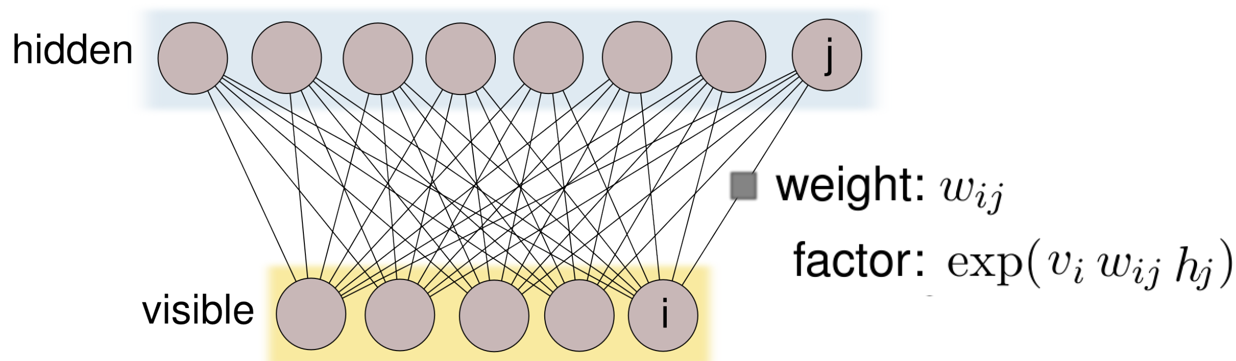
$$\psi_{ij}(x_i, x_j) = \exp(x_i W_{ij} x_j)$$

(In English: higher value of parameter W_{ij} leads to higher correlation between X_i and X_j on value 1)



Restricted Boltzman Machines

- Assume visible units are one layer, and hidden units are another.
- Throw out all the connections within each layer.



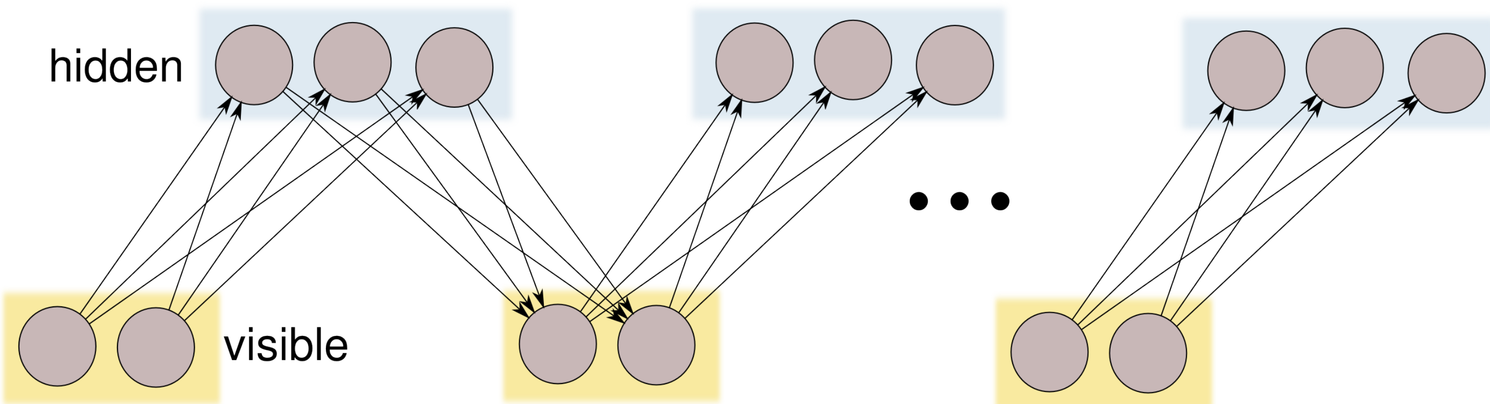
- $h_j \perp\!\!\!\perp h_k \mid \mathbf{v}$
- the posterior $P(\mathbf{h} \mid \mathbf{v})$ factors
  c.f. in a belief net, the *prior* $P(\mathbf{h})$ factors
- no explaining away

Slide from Marcus Frean, MLSS Tutorial 2010

Restricted Boltzman Machines

Alternating Gibbs sampling

Since none of the units within a layer are interconnected, we can do Gibbs sampling by updating the whole layer at a time.

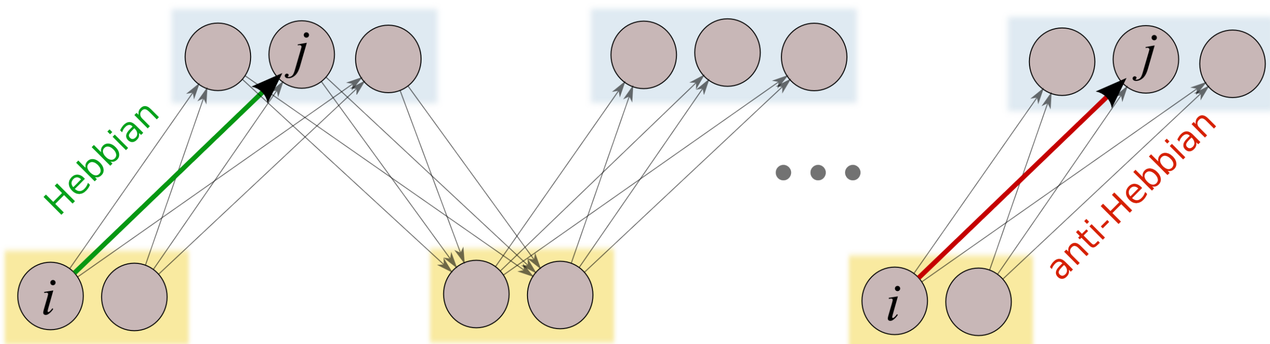


(with time running from left $\longrightarrow$ right)

Slide from Marcus Frean, MLSS Tutorial 2010

Restricted Boltzman Machines

learning in an RBM



Repeat for all data:

1. start with a training vector on the visible units
2. then alternate between updating all the hidden units in parallel and updating all the visible units in parallel

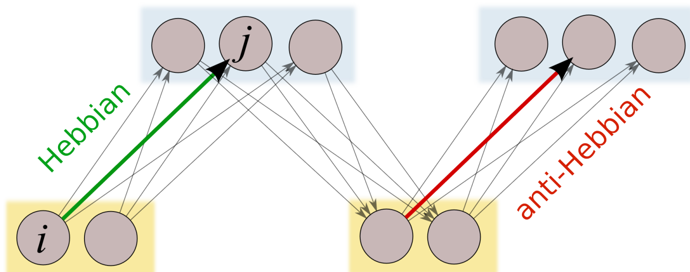
$$\Delta w_{ij} = \eta \left[\langle v_i \, h_j \rangle^0 - \langle v_i \, h_j \rangle^\infty \right]$$

restricted connectivity is trick #1:

it saves waiting for equilibrium in the clamped phase.

Restricted Boltzman Machines

trick # 2: curtail the Markov chain during learning



Repeat for all data:

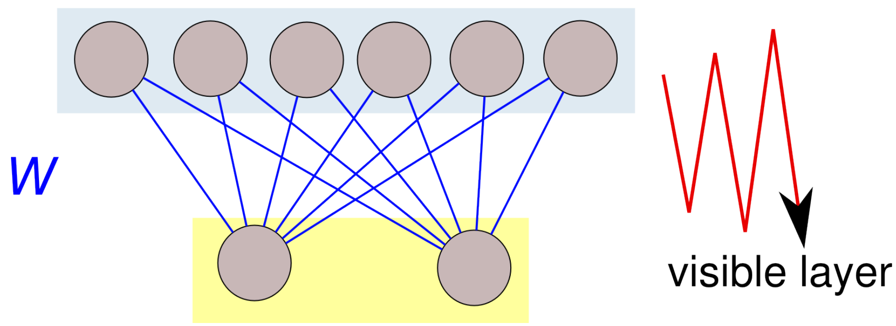
1. start with a training vector on the visible units
2. update all the hidden units in parallel
3. update all the visible units in parallel to get a "reconstruction"
4. update the hidden units again

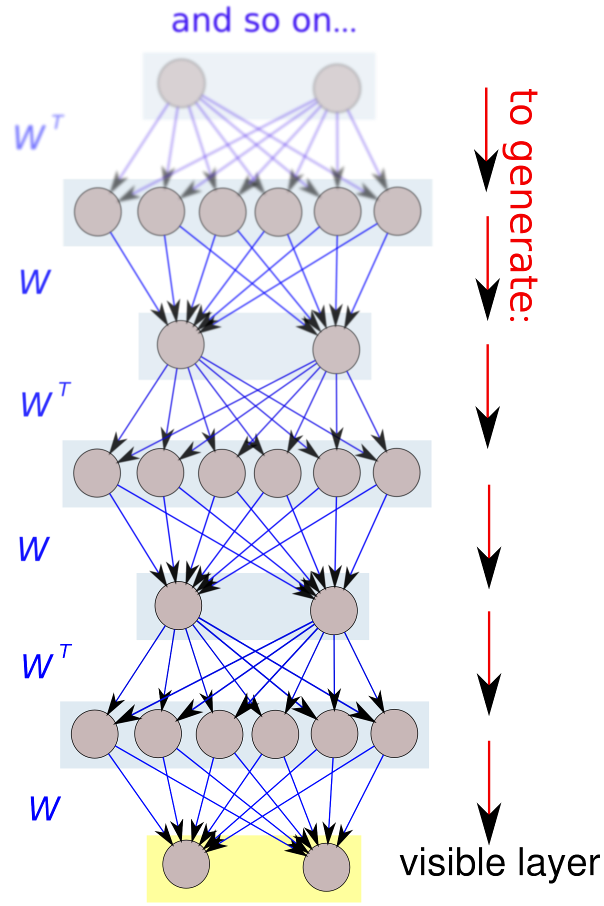
$$\Delta w_{ij} = \eta \left[\langle v_i \, h_j \rangle^0 - \langle v_i \, h_j \rangle^1 \right]$$

This is not following the correct gradient, but works well in practice. Geoff Hinton calls it learning by "contrastive divergence".

Deep Belief Networks (DBNs)

RBMs are equivalent to infinitely deep belief networks



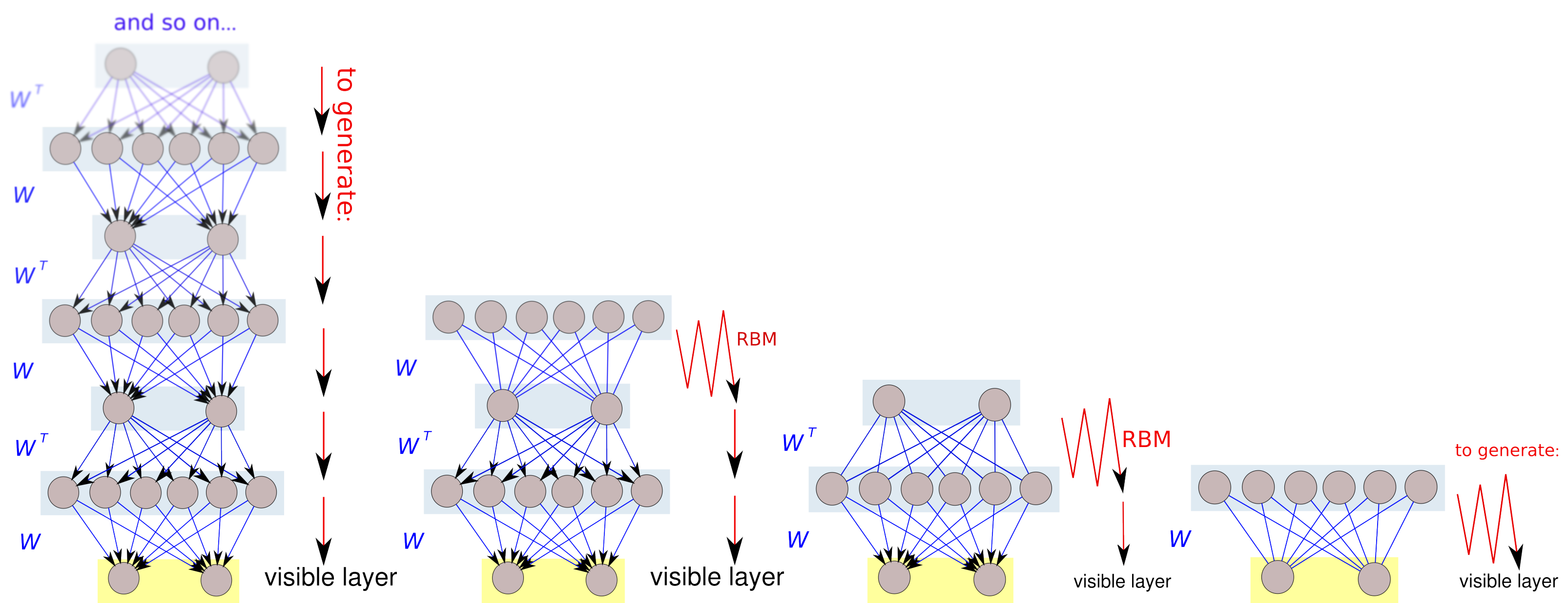


sampling from this is the same as sampling from the network on the right.

Slide from Marcus Frean, MLSS Tutorial 2010

Deep Belief Networks (DBNs)

RBMs are equivalent to infinitely deep belief networks



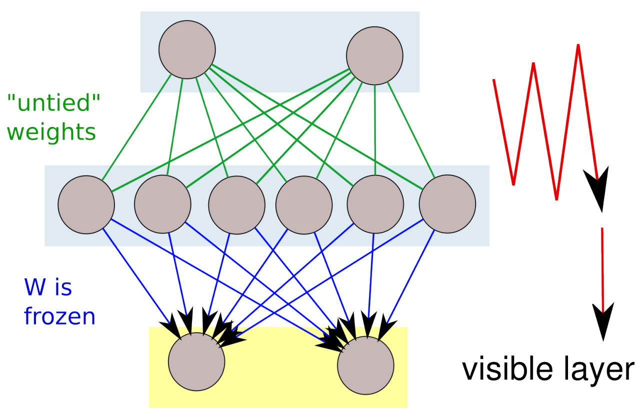
- So when we train an RBM, we're really training an ∞^{ly} deep sigmoid belief net!
- It's just that the weights of all layers are tied.

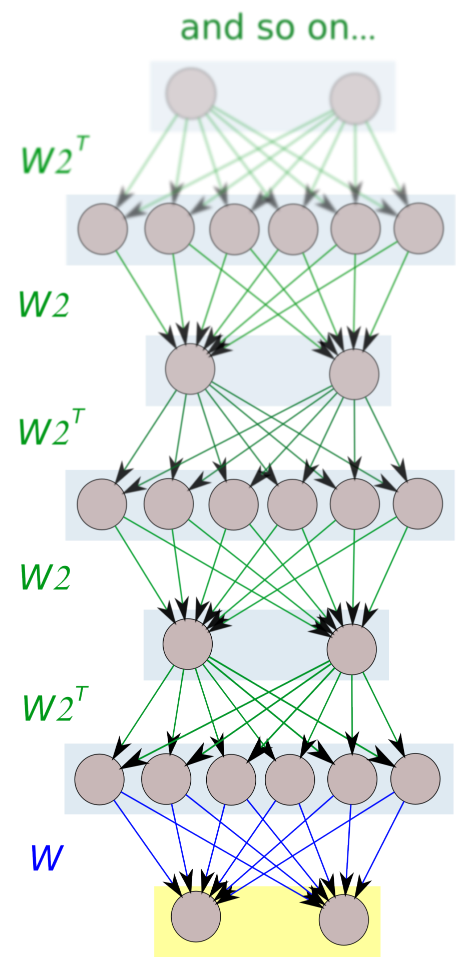
Slide from Marcus Frean, MLSS Tutorial 2010

Deep Belief Networks (DBNs)

Un-tie the weights from layers 2 to infinity

If we freeze the first RBM, and then train another RBM atop it, we are untying the weights of layers 2+ in the ∞ net (which remain tied together).



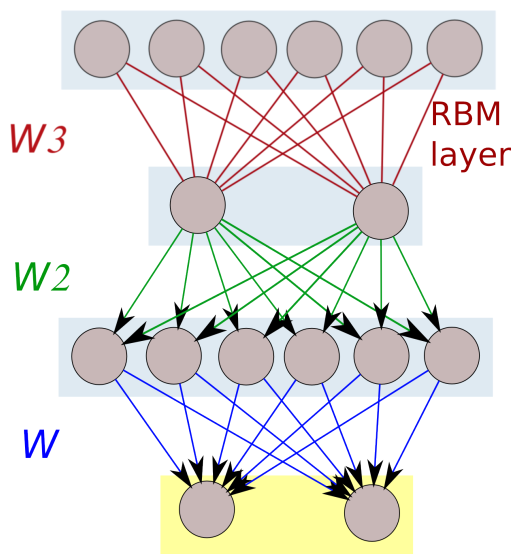


Slide from Marcus Frean, MLSS Tutorial 2010

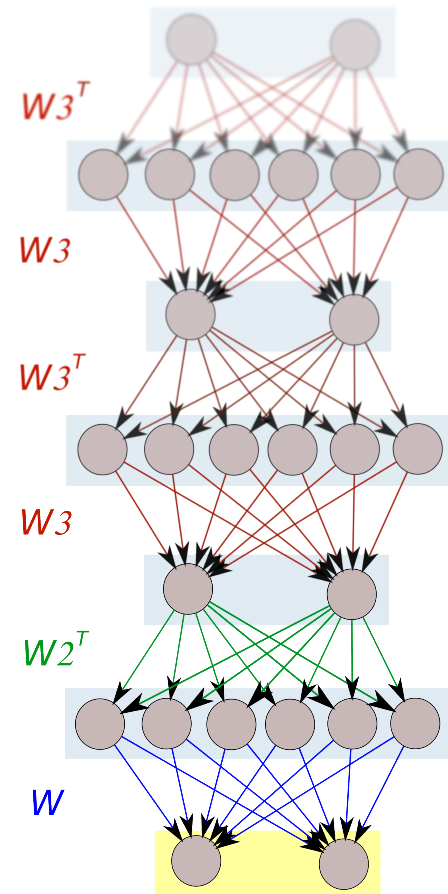
Deep Belief Networks (DBNs)

Un-tie the weights from layers 3 to infinity

and ditto for the 3rd layer...



and so on...



Slide from Marcus Frean, MLSS Tutorial 2010

Deep Belief Networks (DBNs)

fine-tuning with the wake-sleep algorithm

So far, the up and down weights have been symmetric, as required by the Boltzmann machine learning algorithm. And we didn't change the lower levels after "freezing" them.

- wake: do a bottom-up pass, starting with a pattern from the training set. Use the delta rule to make this more likely *under the generative model*.
- sleep: do a top-down pass, starting from an equilibrium sample from the top RBM. Use the delta rule to make this more likely *under the the recognition model*.

[CD version: start top RBM at the sample from the wake phase, and don't wait for equilibrium before doing the top-down pass].

wake-sleep learning algorithm

unties the recognition weights from the generative ones

Slide from Marcus Frean, MLSS Tutorial 2010

Unsupervised Learning of DBNs

Setting A: DBN Autoencoder

I. Pre-train a stack of RBMs in greedy layerwise fashion

II. Unroll the RBMs to create an autoencoder (i.e. bottom-up and top-down weights are untied)

III. Fine-tune the parameters using backpropagation

Figure from (Hinton & Salakhutinov, 2006)

Unsupervised Learning of DBNs

Setting A: DBN Autoencoder

I. Pre-train a stack of RBMs in greedy layerwise fashion

II. Unroll the RBMs to create an autoencoder (i.e. bottom-up and top-down weights are untied)

III. Fine-tune the parameters using backpropagation

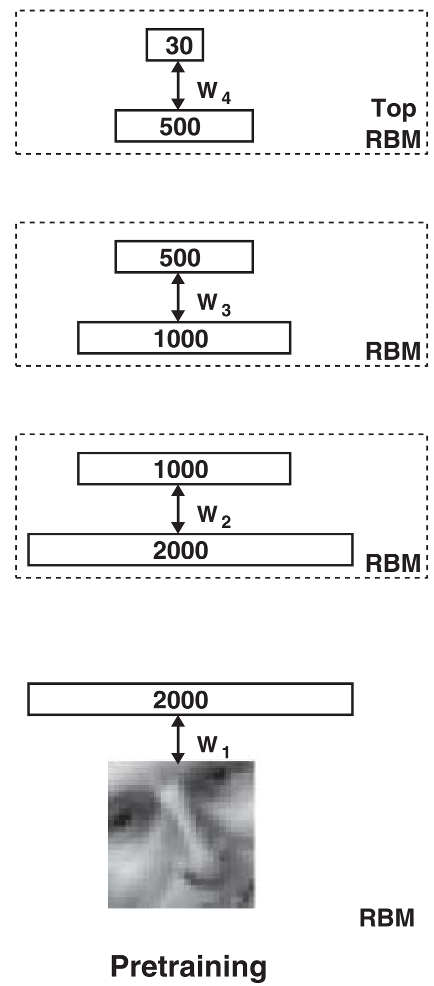


Figure from (Hinton & Salakhutinov, 2006)

Unsupervised Learning of DBNs

Setting A: DBN Autoencoder

I. Pre-train a stack of RBMs in greedy layerwise fashion

II. Unroll the RBMs to create an autoencoder (i.e. bottom-up and top-down weights are untied)

III. Fine-tune the parameters using backpropagation

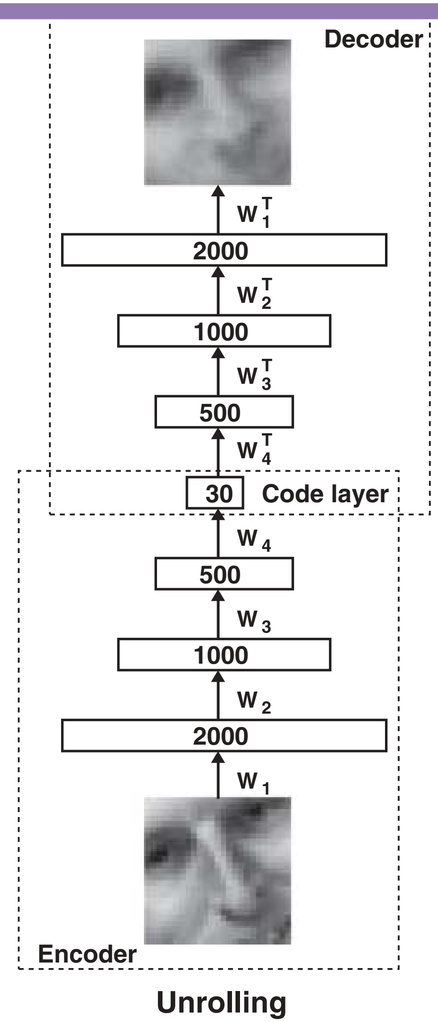


Figure from (Hinton & Salakhutinov, 2006)

Unsupervised Learning of DBNs

Setting A: DBN Autoencoder

I. Pre-train a stack of RBMs in greedy layerwise fashion

II. Unroll the RBMs to create an autoencoder (i.e. bottom-up and top-down weights are untied)

III. Fine-tune the parameters using backpropagation

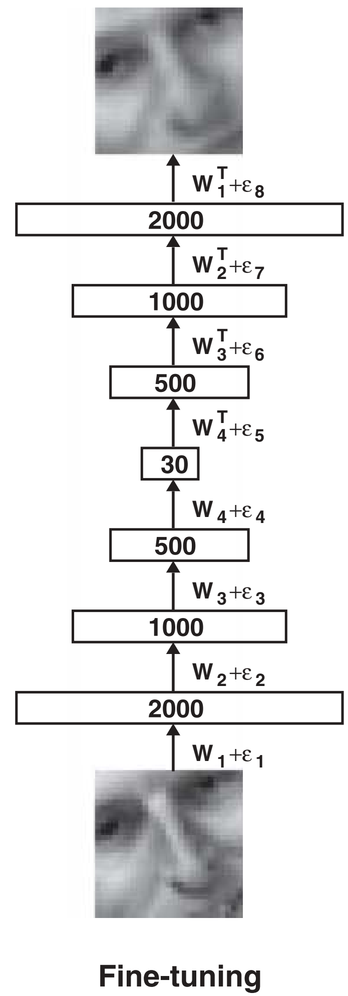


Figure from (Hinton & Salakhutinov, 2006)

Supervised Learning of DBNs

Setting B: DBN classifier

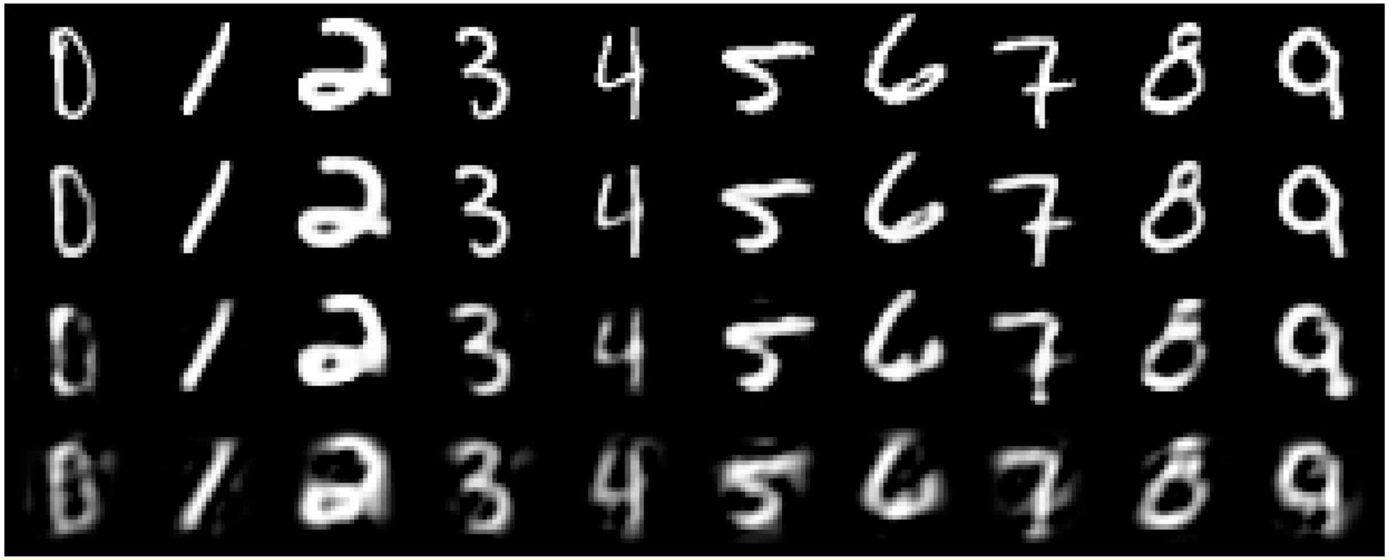
I. Pre-train a stack of RBMs in greedy layerwise fashion (unsupervised)

II. Fine-tune the parameters using backpropagation by minimizing classification error on the training data

Figure from (Hinton & Salakhutinov, 2006)

DBNs

MNIST Digit **Generation**



- Comparison of deep autoencoder, logistic PCA, and PCA
- Each method projects the *real data* down to a vector of 30 real numbers
- Then reconstructs the data from the low-dimensional projection

Figure from Hinton, NIPS Tutorial 2007

Learning Deep Belief Networks (DBNs)

Setting B: DBN Autoencoder

I. Pre-train a stack of RBMs in greedy layerwise fashion
II. Unroll the RBMs to create an autoencoder (i.e. bottom-up and top-down weights are untied)
III. Fine-tune the parameters using backpropagation

Figure from (Hinton & Salakhutinov, 2006)

DBNs

MNIST Digit **Generation**

- This section: Suppose you want to build a **generative** model capable of explaining handwritten digits
- Goal:
  - To have a model p(x) **from which we can sample digits** that look realistic
  - Learn **unsupervised** hidden representation of an image

Figure 8: Each row shows 10 samples from the generative model with a particular label clamped on. The top-level associative memory is run for 1000 iterations of alternating Gibbs sampling between samples.

Samples from a DBN trained on MNIST

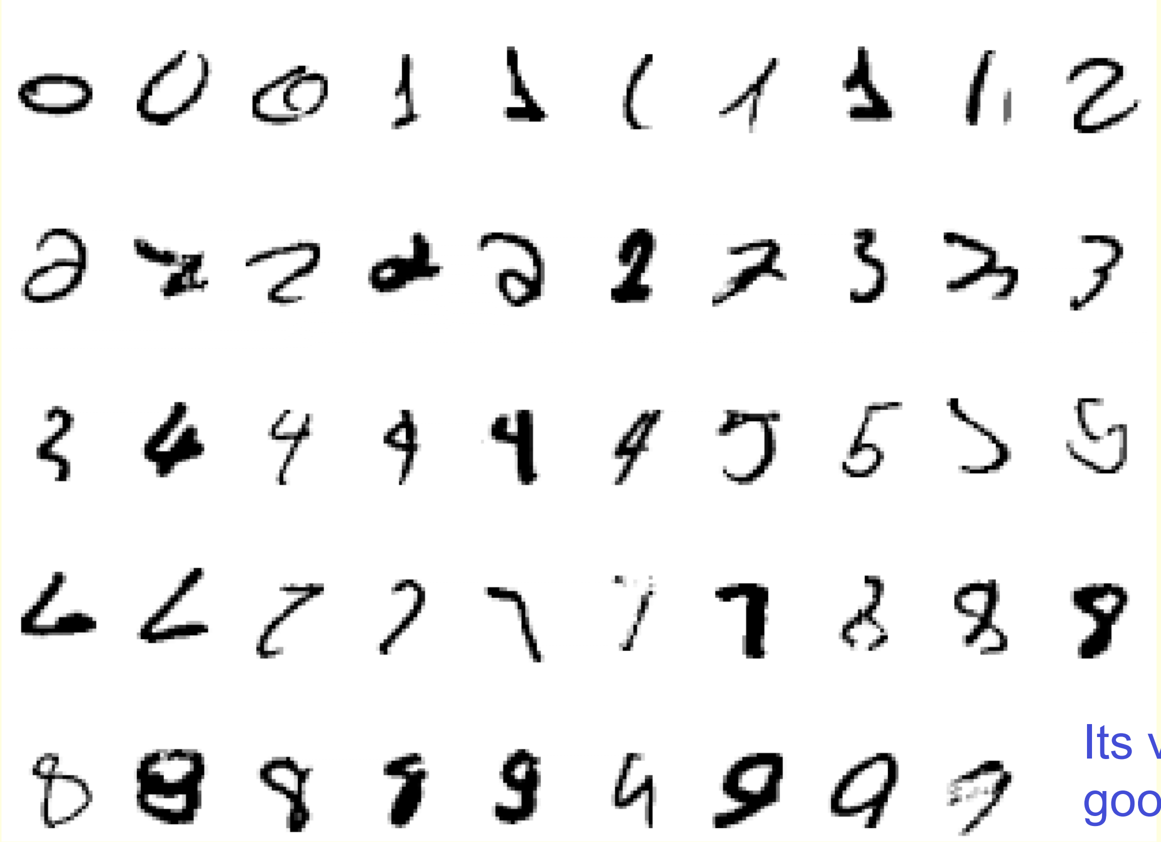
Figure from (Hinton et al., 2006)

DBNs

MNIST Digit **Recognition**

Experimental evaluation of DBN with greedy layer-wise pre-training and fine-tuning via the wake-sleep algorithm

Examples of correctly recognized handwritten digits that the neural network had never seen before



Its very good

Slide from Hinton, NIPS Tutorial 2007

MNIST Digit **Recognition**

Experimental evaluation of DBN with greedy layer-wise pre-training and fine-tuning via the wake-sleep algorithm

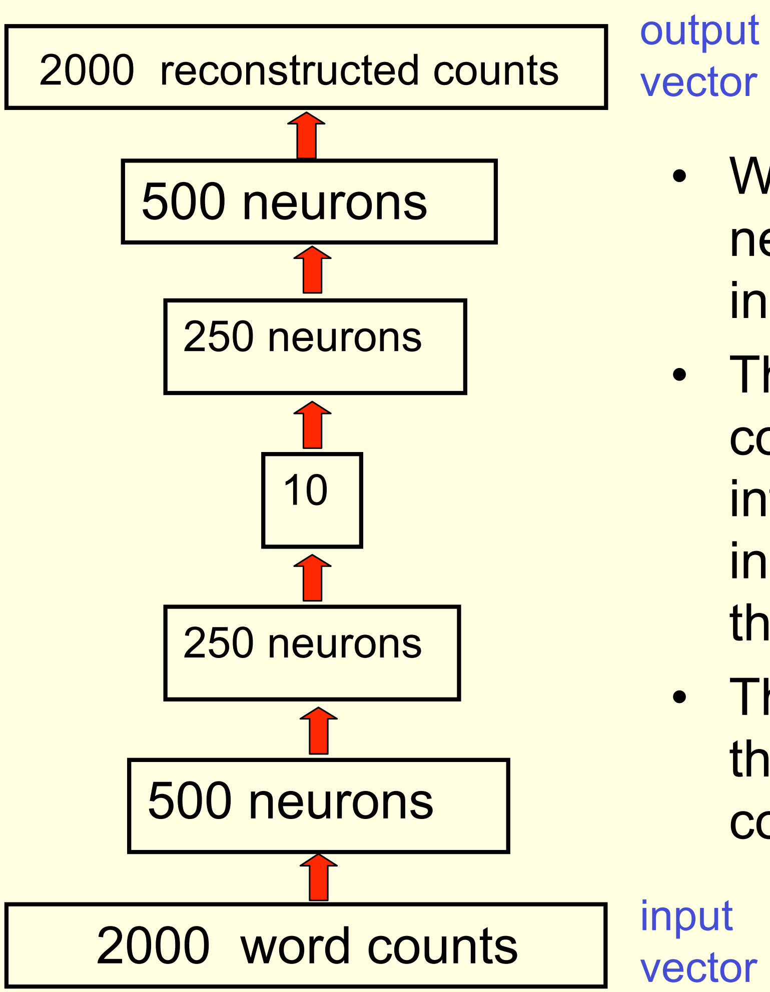
How well does it discriminate on MNIST test set with no extra information about geometric distortions?

• Generative model based on RBM's	1.25%
• Support Vector Machine (Decoste et. al.)	1.4%
• Backprop with 1000 hiddens (Platt)	~1.6%
• Backprop with 500 -->300 hiddens	~1.6%
• K-Nearest Neighbor	~ 3.3%
• See Le Cun et. al. 1998 for more results	

- Its better than backprop and much more neurally plausible because the neurons only need to send one kind of signal, and the teacher can be another sensory input.

Slide from Hinton, NIPS Tutorial 2007

Document Clustering and Retrieval



- We train the neural network to reproduce its input vector as its output
- This forces it to compress as much information as possible into the 10 numbers in the central bottleneck.
- These 10 numbers are then a good way to compare documents.

Slide from Hinton, NIPS Tutorial 2007

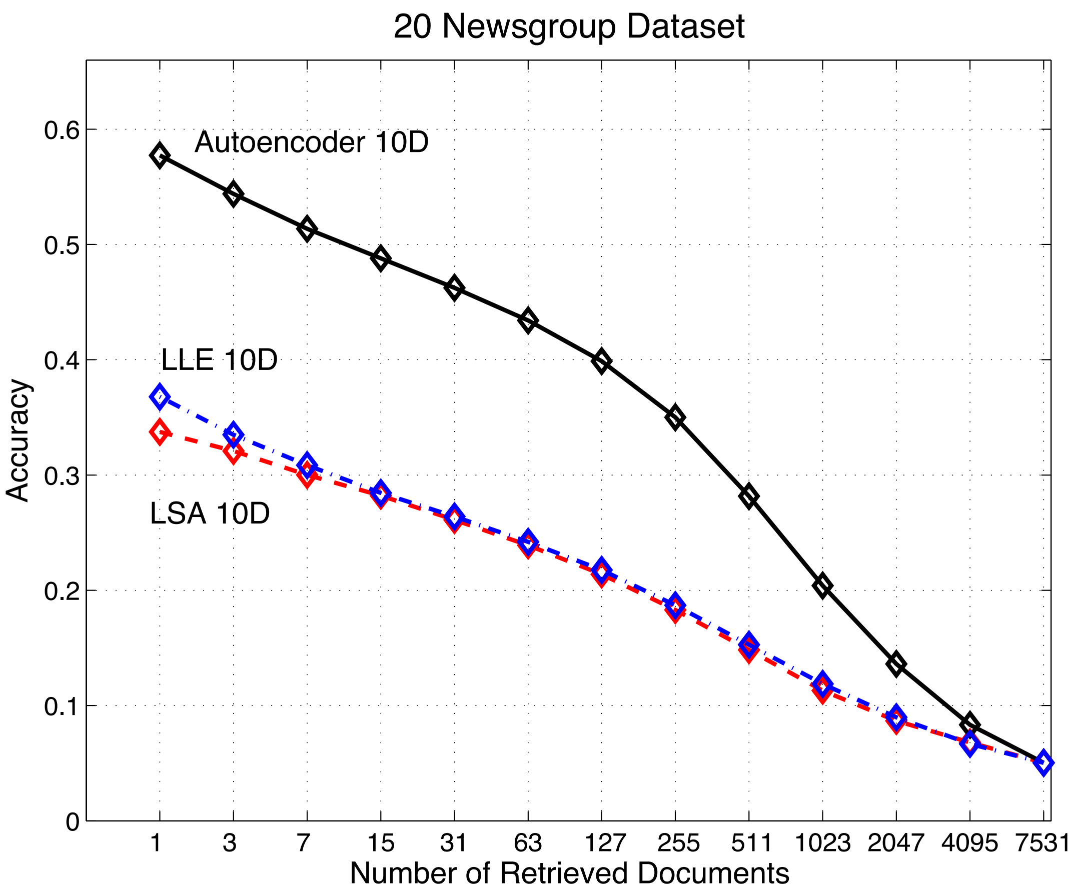
Document Clustering and Retrieval

Performance of the autoencoder at document retrieval

- Train on bags of 2000 words for 400,000 training cases of business documents.
  - First train a stack of RBM's. Then fine-tune with backprop.
- Test on a separate 400,000 documents.
  - Pick one test document as a query. Rank order all the other test documents by using the cosine of the angle between codes.
  - Repeat this using each of the 400,000 test documents as the query (requires 0.16 trillion comparisons).
- Plot the number of retrieved documents against the proportion that are in the same hand-labeled class as the query document.

Slide from Hinton, NIPS Tutorial 2007

Document Clustering and Retrieval



Retrieval Results

- Goal: given a query document, retrieve the relevant test documents
- Figure shows accuracy for varying numbers of retrieved test docs

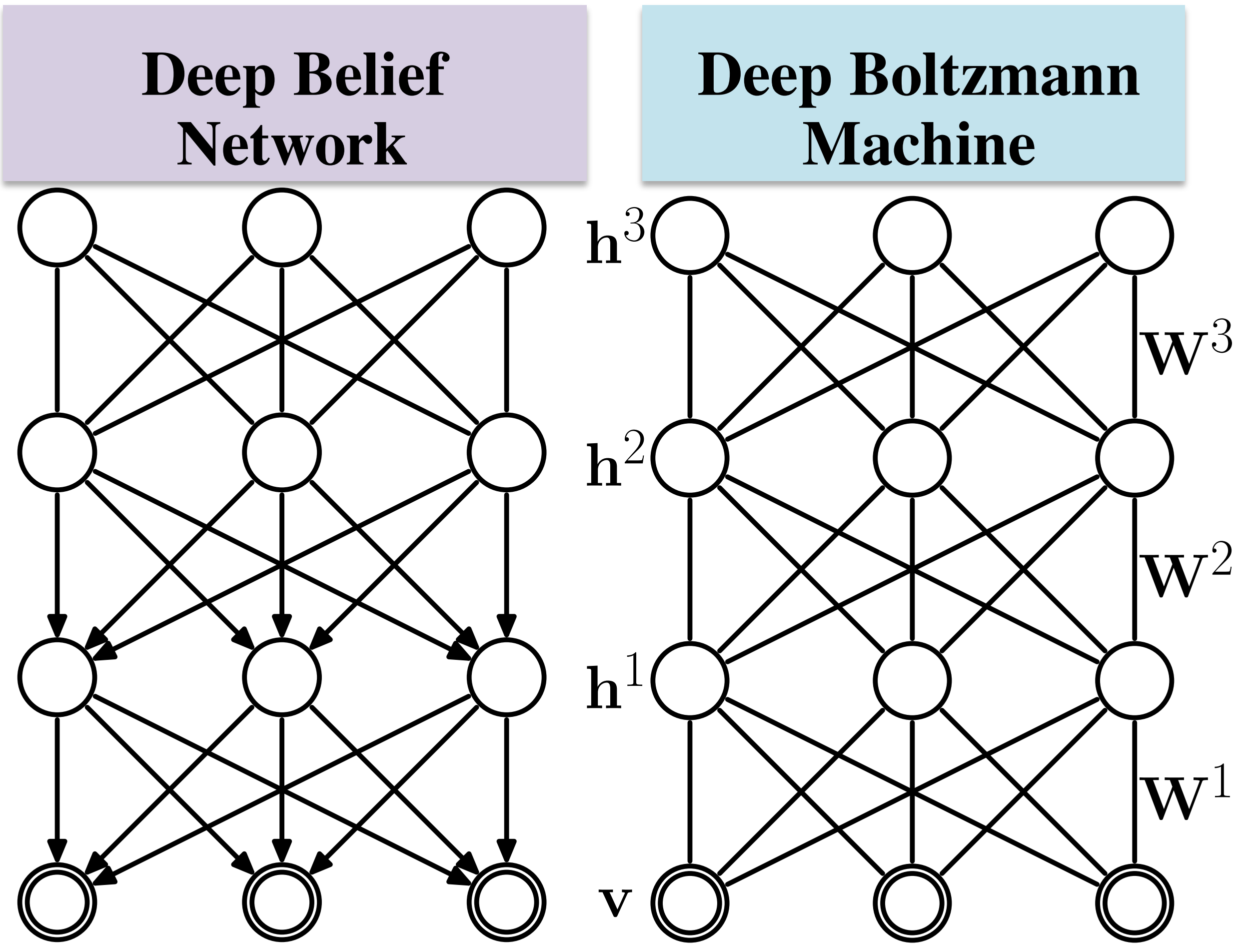
Figure from (Hinton and Salakhutdinov, 2006)

Outline

- Motivation
- Deep Neural Networks (DNNs)
  - Background: Decision functions
  - Background: Neural Networks
  - Three ideas for training a DNN
  - Experiments: MNIST digit classification
- Deep Belief Networks (DBNs)
  - Sigmoid Belief Network
  - Contrastive Divergence learning
  - Restricted Boltzman Machines (RBMs)
  - RBMs as infinitely deep Sigmoid Belief Nets
  - Learning DBNs
- **Deep Boltzman Machines (DBMs)**
  - Boltzman Machines
  - Learning Boltzman Machines
  - Learning DBMs

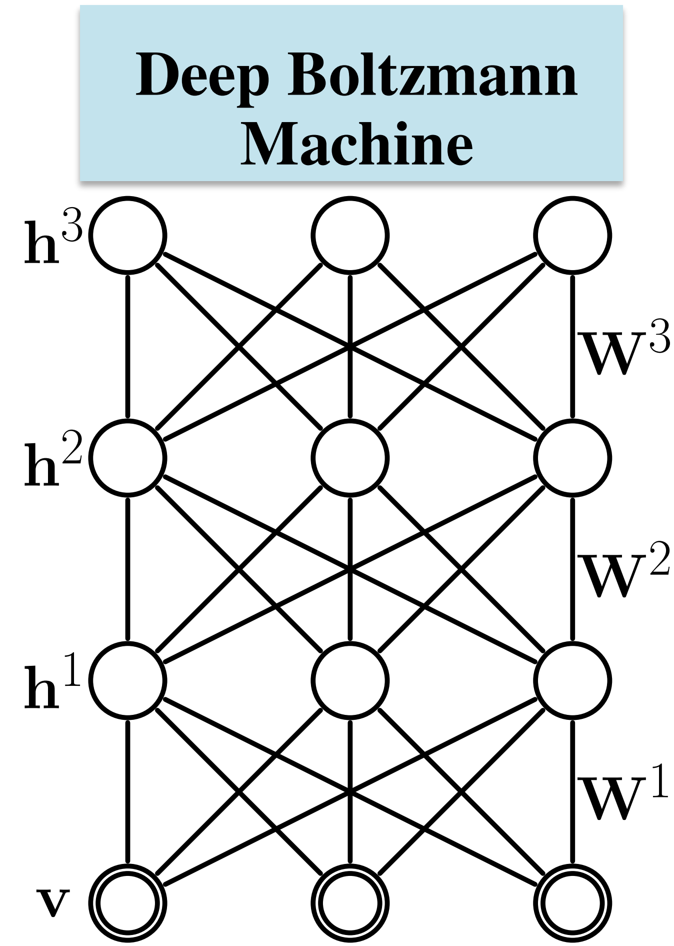
Deep Boltzman Machines

- DBNs are a hybrid directed/ undirected graphical model
- DBMs are a purely undirected graphical model



Deep Boltzman Machines

Can we use the same techniques to train a DBM?

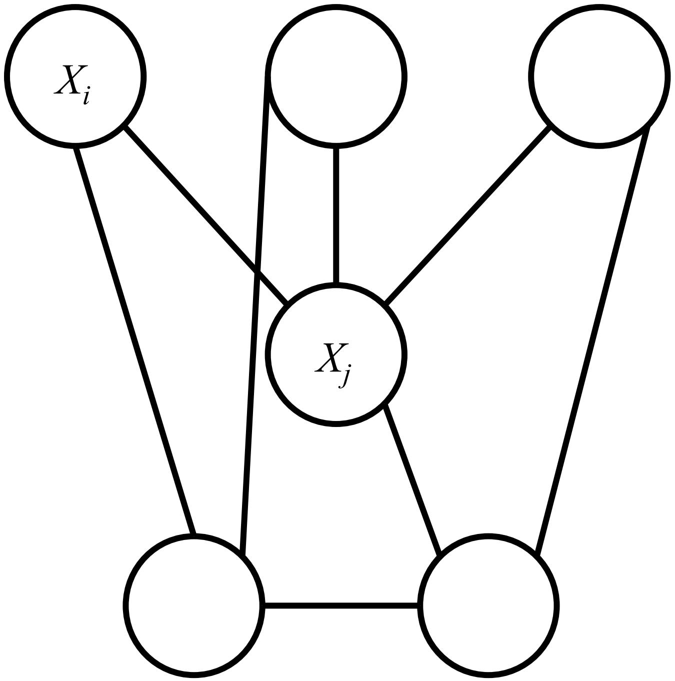


Learning Standard Boltzman Machines

- Undirected graphical model of binary variables with pairwise potentials
- Parameterization of the potentials:

$$\psi_{ij}(x_i, x_j) = \exp(x_i W_{ij} x_j)$$

(In English: higher value of parameter W_{ij} leads to higher correlation between X_i and X_j on value 1)



DBMs

Learning Standard Boltzman Machines

Visible units: $\mathbf{v} \in \{0,1\}^D$

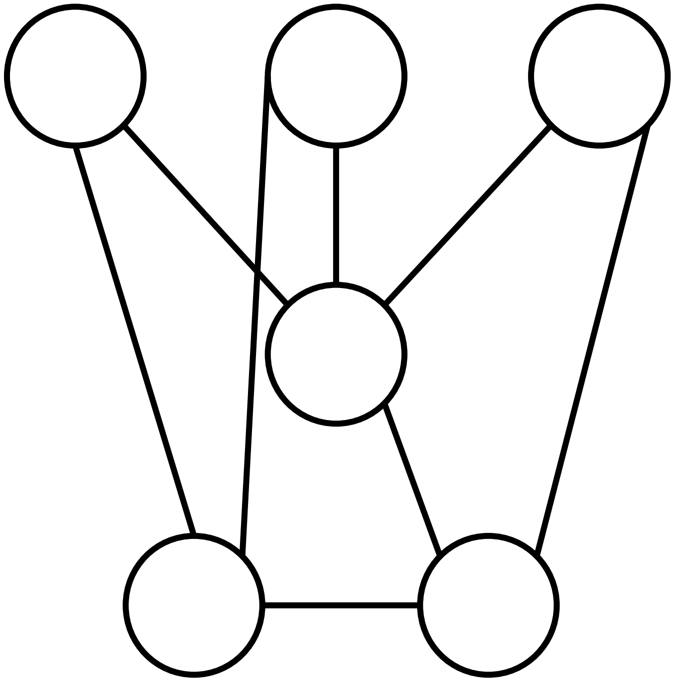
Hidden units: $\mathbf{h} \in \{0,1\}^P$

Likelihood:

$$E(\mathbf{v},\mathbf{h};\theta) = -\frac{1}{2}\mathbf{v}^\top \mathbf{L}\mathbf{v} - \frac{1}{2}\mathbf{h}^\top \mathbf{J}\mathbf{h} - \mathbf{v}^\top \mathbf{W}\mathbf{h},$$

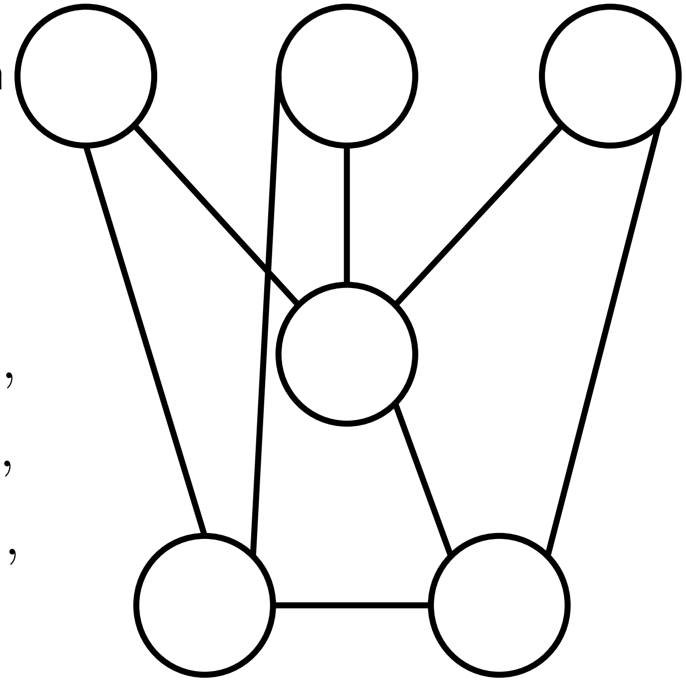
$$p(\mathbf{v};\theta) = \frac{p^*(\mathbf{v};\theta)}{Z(\theta)} = \frac{1}{Z(\theta)} \sum_{h} \exp\left(-E(\mathbf{v},\mathbf{h};\theta)\right),$$

$$Z(\theta) = \sum_{\mathbf{v}} \sum_{\mathbf{h}} \exp\left(-E(\mathbf{v},\mathbf{h};\theta)\right),$$



Learning Standard Boltzman Machines

(Old) idea from Hinton & Sejnowski (1983): For each iteration of optimization, run a separate MCMC chain for each of the data and model expectations to approximate the parameter updates.



Delta updates to each of model parameters:

$$\begin{aligned} \Delta \mathbf{W} &= \alpha \left(\mathrm{E}_{P_{\text{data}}}[\mathbf{v}\mathbf{h}^\top] - \mathrm{E}_{P_{\text{model}}}[\mathbf{v}\mathbf{h}^\top] \right), \\ \Delta \mathbf{L} &= \alpha \left(\mathrm{E}_{P_{\text{data}}}[\mathbf{v}\mathbf{v}^\top] - \mathrm{E}_{P_{\text{model}}}[\mathbf{v}\mathbf{v}^\top] \right), \\ \Delta \mathbf{J} &= \alpha \left(\mathrm{E}_{P_{\text{data}}}[\mathbf{h}\mathbf{h}^\top] - \mathrm{E}_{P_{\text{model}}}[\mathbf{h}\mathbf{h}^\top] \right), \end{aligned}$$

Full conditionals for Gibbs sampler:

$$p(h_j = 1 | \mathbf{v}, \mathbf{h}_{-j}) = \sigma\Big(\sum_{i=1}^{D} W_{ij} v_i + \sum_{m=1 \setminus j}^{P} J_{jm} h_j\Big),$$

$$p(v_i = 1 | \mathbf{h}, \mathbf{v}_{-i}) = \sigma\Big(\sum_{j=1}^{P} W_{ij} h_j + \sum_{k=1 \setminus i}^{D} L_{ik} v_j\Big),$$

Learning Standard Boltzman Machines

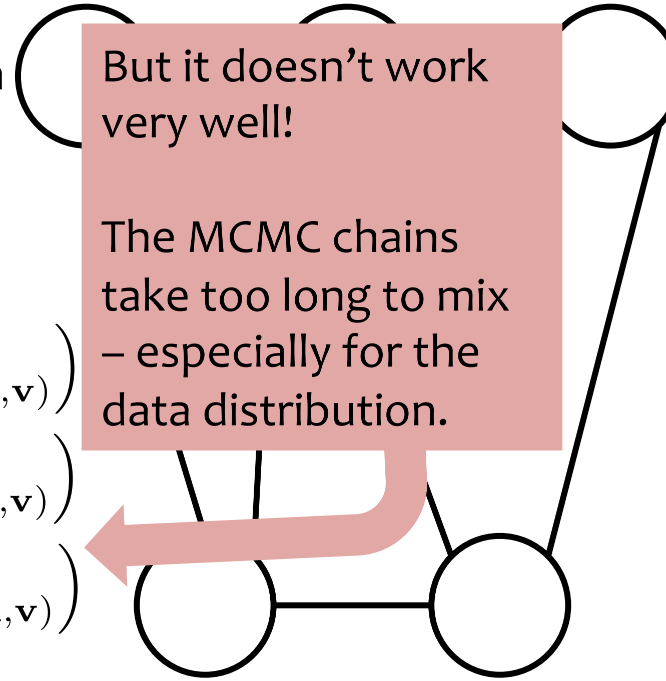
(Old) idea from Hinton & Sejnowski (1983): For each iteration of optimization, run a separate MCMC chain for each of the data and model expectations to approximate the parameter updates.

Delta updates to each of model parameters:

$$\Delta \mathbf{W} = \alpha \left(\langle \mathbf{v}\mathbf{h}^T \rangle_{\mathbf{v} \in \mathcal{D}, \mathbf{h} \sim p(\mathbf{h}|\mathbf{v})} - \langle \mathbf{v}\mathbf{h}^T \rangle_{\mathbf{v}, \mathbf{h} \sim p(\mathbf{h}, \mathbf{v})} \right)$$

$$\Delta \mathbf{L} = \alpha \left(\langle \mathbf{v}\mathbf{v}^T \rangle_{\mathbf{v} \in \mathcal{D}, \mathbf{h} \sim p(\mathbf{h}|\mathbf{v})} - \langle \mathbf{v}\mathbf{v}^T \rangle_{\mathbf{v}, \mathbf{h} \sim p(\mathbf{h}, \mathbf{v})} \right)$$

$$\Delta \mathbf{J} = \alpha \left(\langle \mathbf{h}\mathbf{h}^T \rangle_{\mathbf{v} \in \mathcal{D}, \mathbf{h} \sim p(\mathbf{h}|\mathbf{v})} - \langle \mathbf{h}\mathbf{h}^T \rangle_{\mathbf{v}, \mathbf{h} \sim p(\mathbf{h}, \mathbf{v})} \right)$$



But it doesn't work very well!

The MCMC chains take too long to mix – especially for the data distribution.

Full conditionals for Gibbs sampler:

$$p(h_j = 1 | \mathbf{v}, \mathbf{h}_{-j}) = \sigma\Big(\sum_{i=1}^{D} W_{ij} v_i + \sum_{m=1 \backslash j}^{P} J_{jm} h_j \Big),$$

$$p(v_i = 1 | \mathbf{h}, \mathbf{v}_{-i}) = \sigma\Big(\sum_{j=1}^{P} W_{ij} h_j + \sum_{k=1 \backslash i}^{D} L_{ik} v_j \Big),$$

Learning Standard Boltzman Machines

(New) idea from Salakhutinov & Hinton (2009):

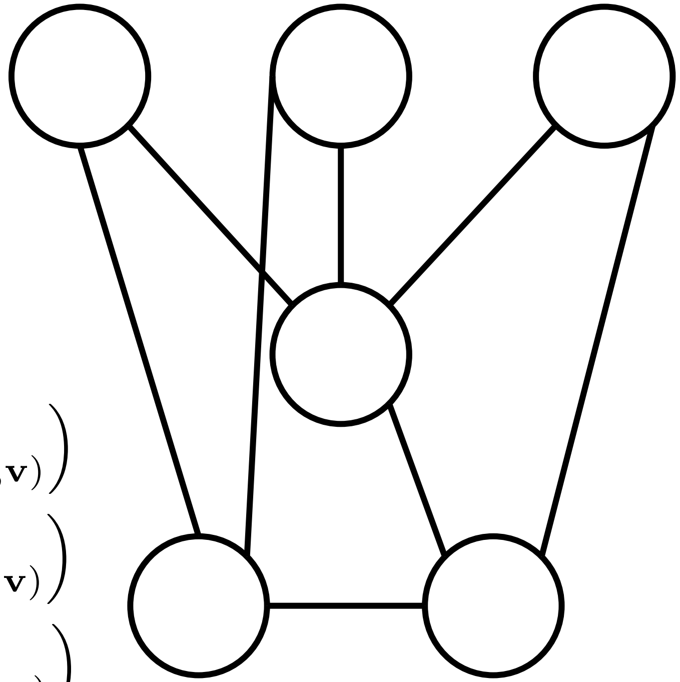
- Step 1) Approximate the data distribution by variational inference.
- Step 2) Approximate the model distribution with a "persistent" Markov chain (from iteration to iteration)

Delta updates to each of model parameters:

$$\Delta \mathbf{W} = \alpha \left(\left\langle \mathbf{v}\mathbf{h}^T \right\rangle_{\mathbf{v} \in \mathcal{D}, \mathbf{h} \sim p(\mathbf{h}|\mathbf{v})} - \left\langle \mathbf{v}\mathbf{h}^T \right\rangle_{\mathbf{v}, \mathbf{h} \sim p(\mathbf{h}, \mathbf{v})} \right)$$

$$\Delta \mathbf{L} = \alpha \left(\left\langle \mathbf{v}\mathbf{v}^T \right\rangle_{\mathbf{v} \in \mathcal{D}, \mathbf{h} \sim p(\mathbf{h}|\mathbf{v})} - \left\langle \mathbf{v}\mathbf{v}^T \right\rangle_{\mathbf{v}, \mathbf{h} \sim p(\mathbf{h}, \mathbf{v})} \right)$$

$$\Delta \mathbf{J} = \alpha \left(\left\langle \mathbf{h}\mathbf{h}^T \right\rangle_{\mathbf{v} \in \mathcal{D}, \mathbf{h} \sim p(\mathbf{h}|\mathbf{v})} - \left\langle \mathbf{h}\mathbf{h}^T \right\rangle_{\mathbf{v}, \mathbf{h} \sim p(\mathbf{h}, \mathbf{v})} \right)$$

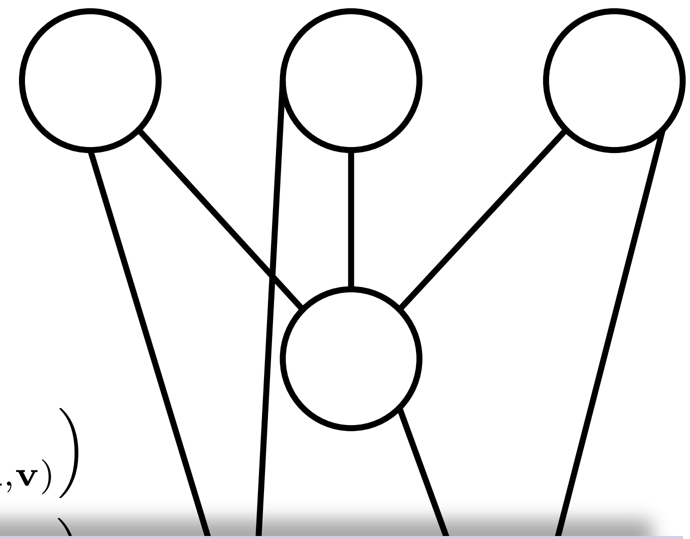


DBMs

Learning Standard Boltzman Machines

(New) idea from Salakhutinov & Hinton (2009):

- Step 1) Approximate the data distribution by variational inference.
- Step 2) Approximate the model distribution with a “persistent” Markov chain (from iteration to iteration)



Delta updates to each of model parameters:

$$\Delta \mathbf{W} = \alpha \left(\langle \mathbf{v}\mathbf{h}^T \rangle_{\mathbf{v} \in \mathcal{D}, \mathbf{h} \sim p(\mathbf{h}|\mathbf{v})} - \langle \mathbf{v}\mathbf{h}^T \rangle_{\mathbf{v}, \mathbf{h} \sim p(\mathbf{h},\mathbf{v})} \right)$$

Step 1) Approximate the data distribution…

Mean-field approximation:

$$q(\mathbf{h}; \mu) = \prod_{j=1}^{P} q(h_i)$$

$$q(h_i = 1) = \mu_i$$

Variational lower-bound of log-likelihood:

$$\ln p(\mathbf{v}; \theta) \geq \sum_{\mathbf{h}} q(\mathbf{h}|\mathbf{v}; \mu) \ln p(\mathbf{v}, \mathbf{h}; \theta) + \mathcal{H}(q)$$

Fixed-point equations for variational params:

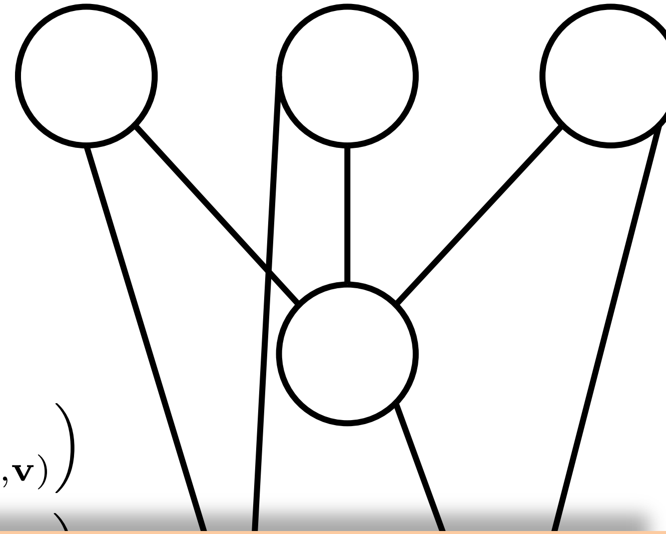
$$\mu_j \leftarrow \sigma\left(\sum_i W_{ij} v_i + \sum_{m \setminus j} J_{mj} \mu_m \right)$$

DBMs

Learning Standard Boltzman Machines

(New) idea from Salakhutinov & Hinton (2009):

- Step 1) Approximate the data distribution by variational inference.
- Step 2) Approximate the model distribution with a "persistent" Markov chain (from iteration to iteration)



Delta updates to each of model parameters:

$$\Delta \mathbf{W} = \alpha \left(\langle \mathbf{v}\mathbf{h}^T \rangle_{\mathbf{v} \in \mathcal{D}, \mathbf{h} \sim p(\mathbf{h}|\mathbf{v})} - \langle \mathbf{v}\mathbf{h}^T \rangle_{\mathbf{v}, \mathbf{h} \sim p(\mathbf{h}, \mathbf{v})} \right)$$

Step 2) Approximate the model distribution…

Why not use variational inference for the model expectation as well?

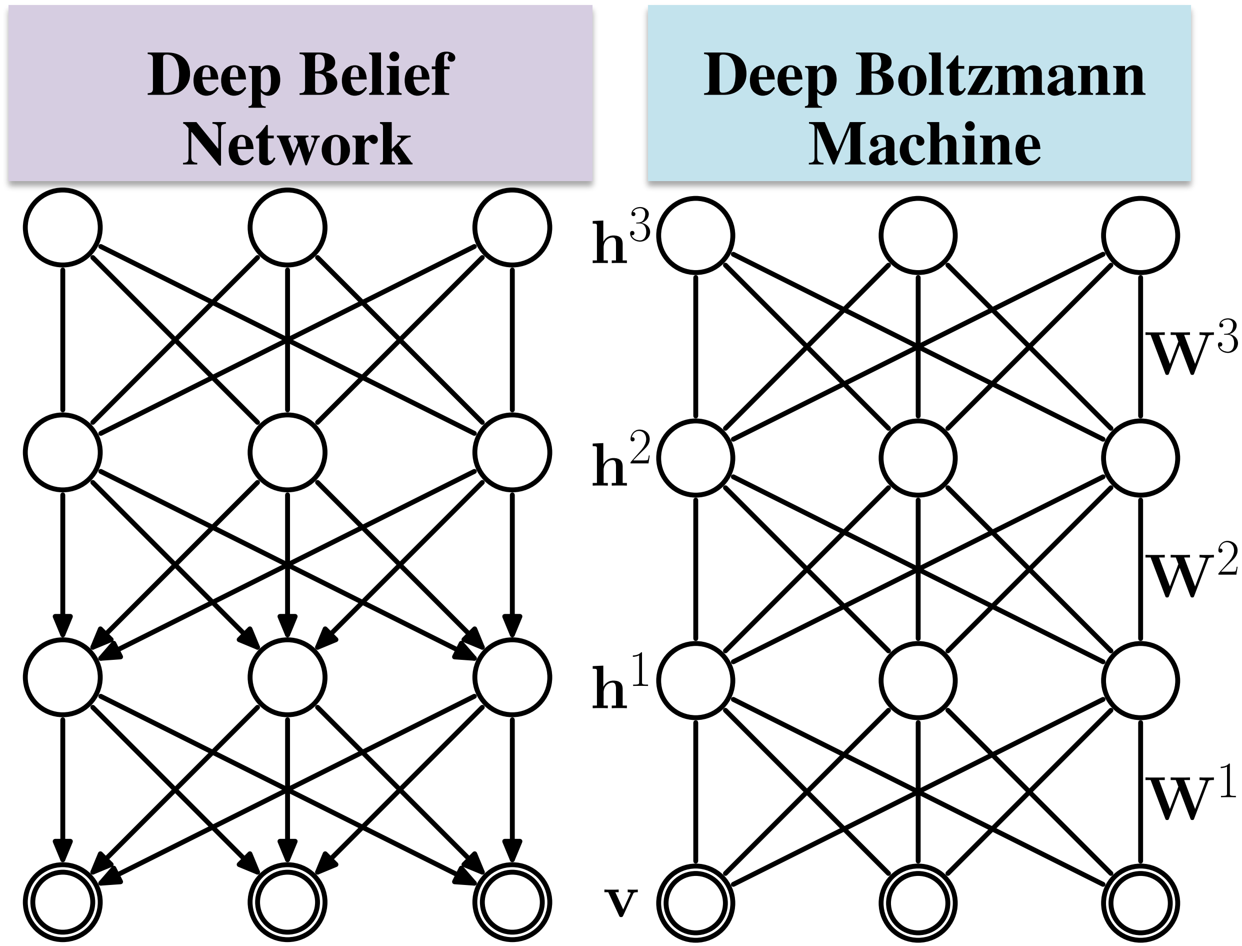
Difference of the two mean-field approximated expectations above would cause learning algorithm to **maximize** divergence between true and mean-field distributions.

Persistent CD adds correlations between successive iterations, but not an issue.

DBMs

Deep Boltzman Machines

- DBNs are a hybrid directed/ undirected graphical model
- DBMs are a purely undirected graphical model

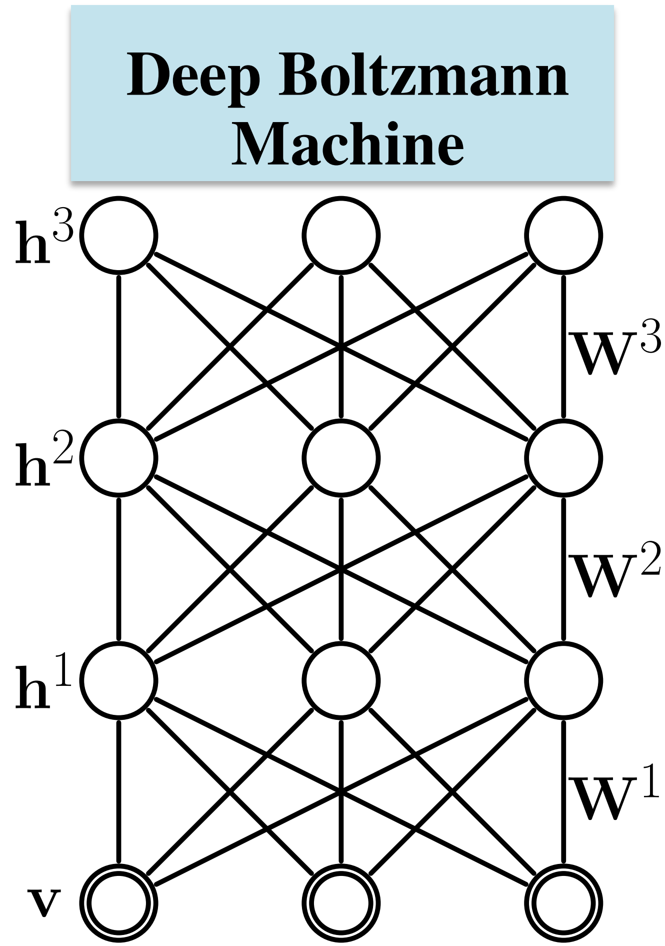


Learning Deep Boltzman Machines

Can we use the same techniques to train a DBM?

I. Pre-train a stack of RBMs in greedy layerwise fashion (requires some caution to avoid double counting)

II. Use those parameters to initialize two step mean-field approach to learning full Boltzman machine (i.e. the full DBM)



Document Clustering and Retrieval

Clustering Results

- Goal: cluster related documents
- Figures show projection to 2 dimensions
- Color shows true categories

PCA

DBN

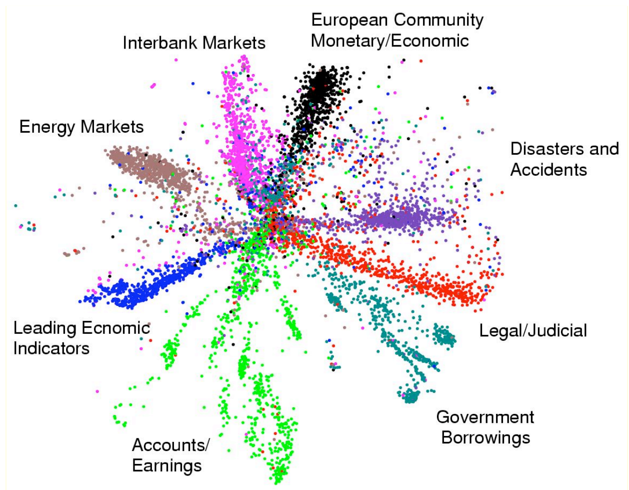


Figure from (Salakhutdinov and Hinton, 2009)

Deep Learning

Lots to explore:

- Other nonlinear functions
  - Rectified Linear Units (ReLUs)
- Popular (classic) architectures:
  - Convolutional Neural Networks (CNN)
  - Long-term Short-term Memory (LSTM)
- Modern architectures
  - Stacked SVMs with random projections
  - Sum-product Networks