School of Computer Science
Carnegie Mellon

10-701 Introduction to Machine Learning

Deep Learning

Matt Gormley
Lecture 13
October 19, 2016

Readings:
Bishop Ch. 4.1.7, Ch. 5
Murphy Ch. 16.5, Ch. 28
Mitchell Ch. 4

Reminders

- Homework 3:
  - due 10/24/16

Outline

- **Deep Neural Networks (DNNs)**
  – Three ideas for training a DNN
  – Experiments: MNIST digit classification
  – Autoencoders
  – Pretraining
- **Recurrent Neural Networks (RNNs)**
  – Bidirectional RNNs
  – Deep Bidirectional RNNs
  – Deep Bidirectional LSTMs
  – Connection to forward-backward algorithm
- **Convolutional Neural Networks (CNNs)**
  – Convolutional layers
  – Pooling layers
  – Image recognition

PRE-TRAINING FOR DEEP NETS

Goals for Today's Lecture

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

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)

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

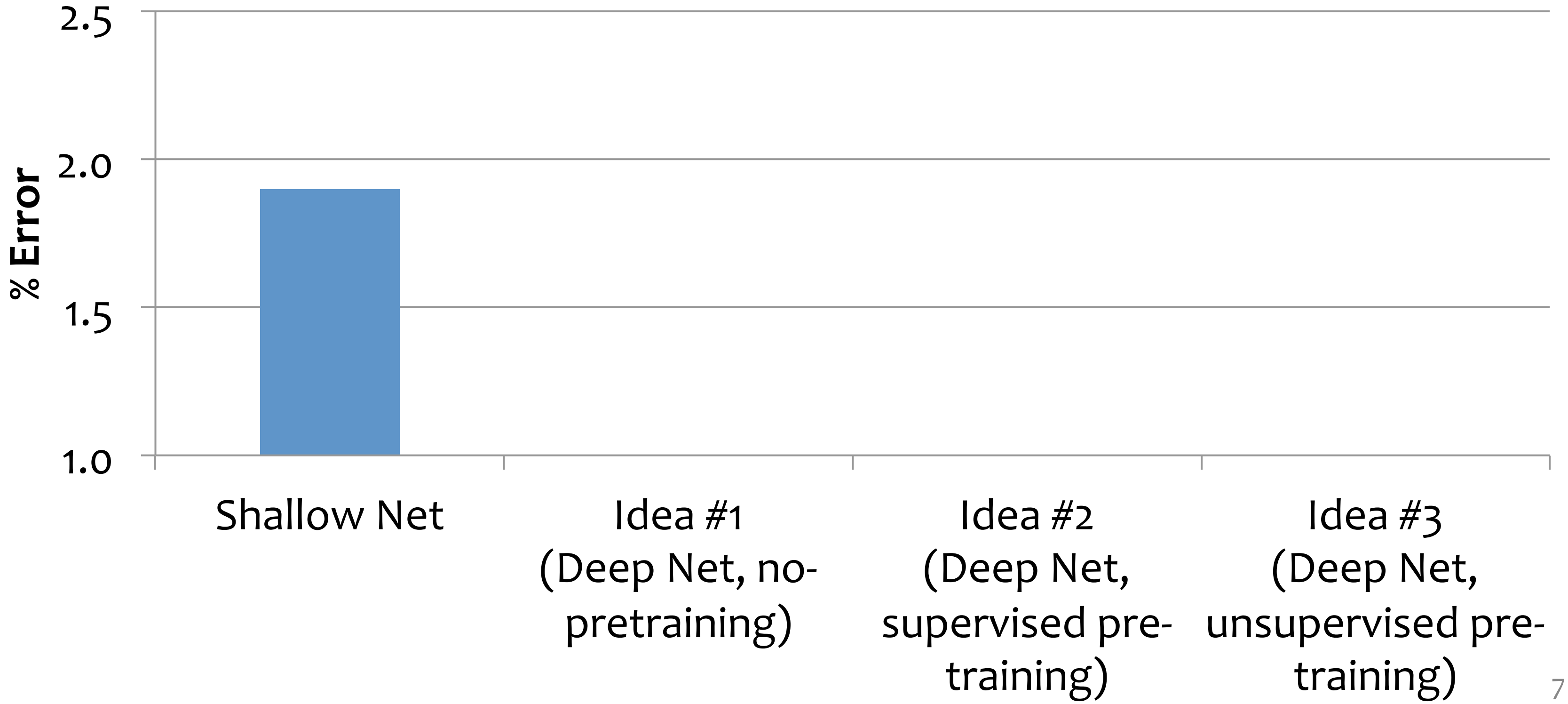
Training

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)

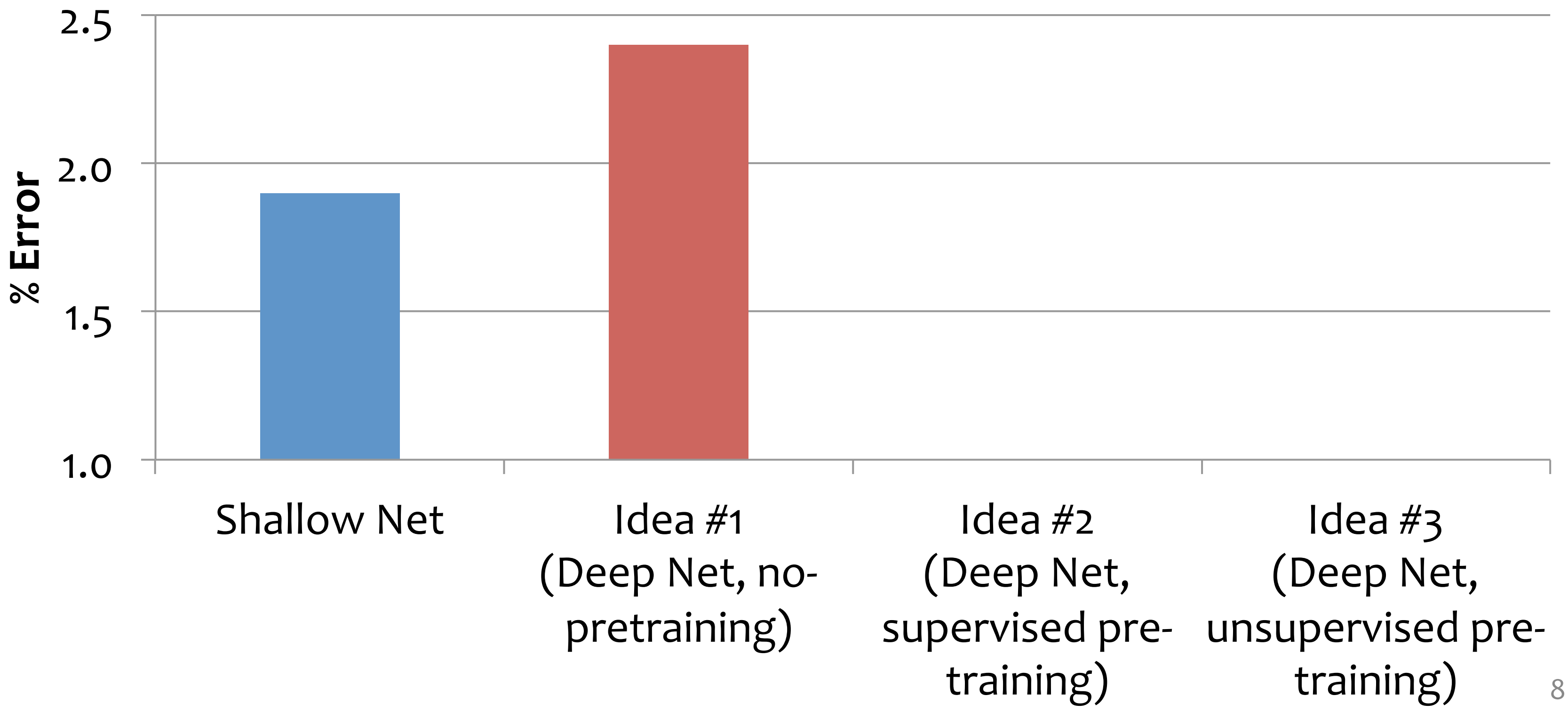
Comparison on MNIST

- 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

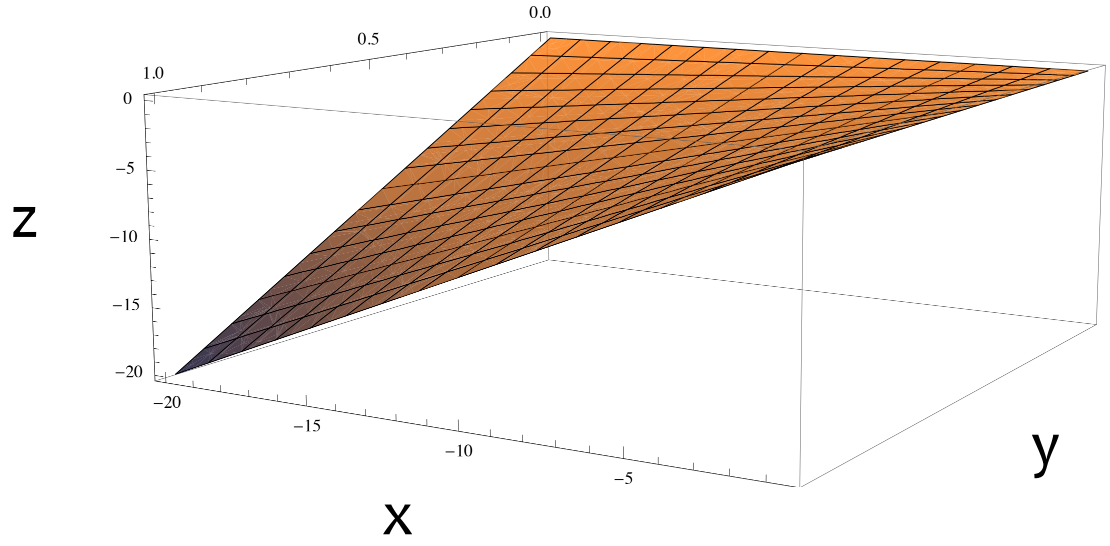
- **Idea #1: (Just like a shallow network)**
  - Compute the supervised gradient by backpropagation.
  - Take small steps in the direction of the gradient (SGD)

- 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*

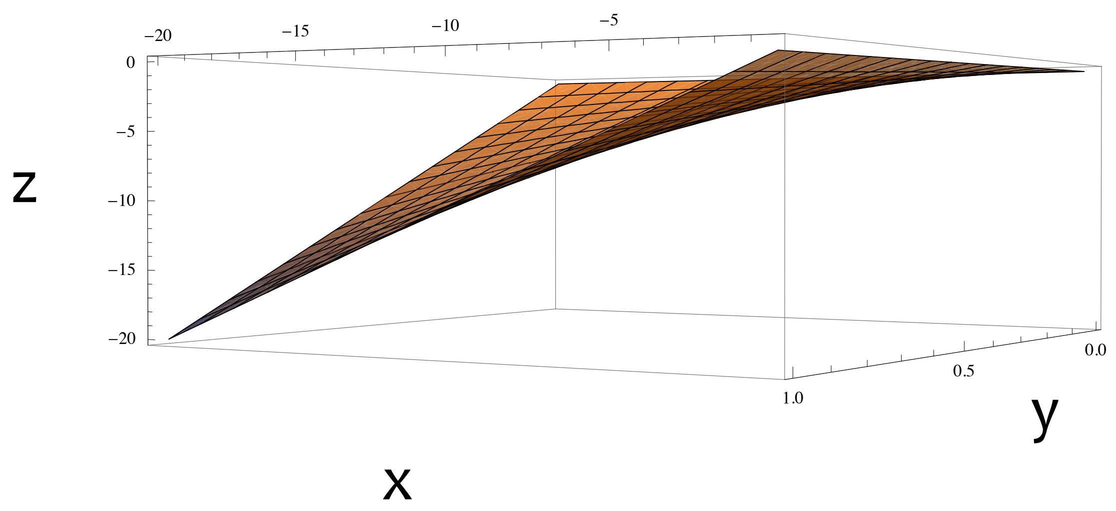
- Where does the nonconvexity come from?
- Even a simple quadratic z = xy objective is nonconvex:



Training

Problem A: *Nonconvexity*

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

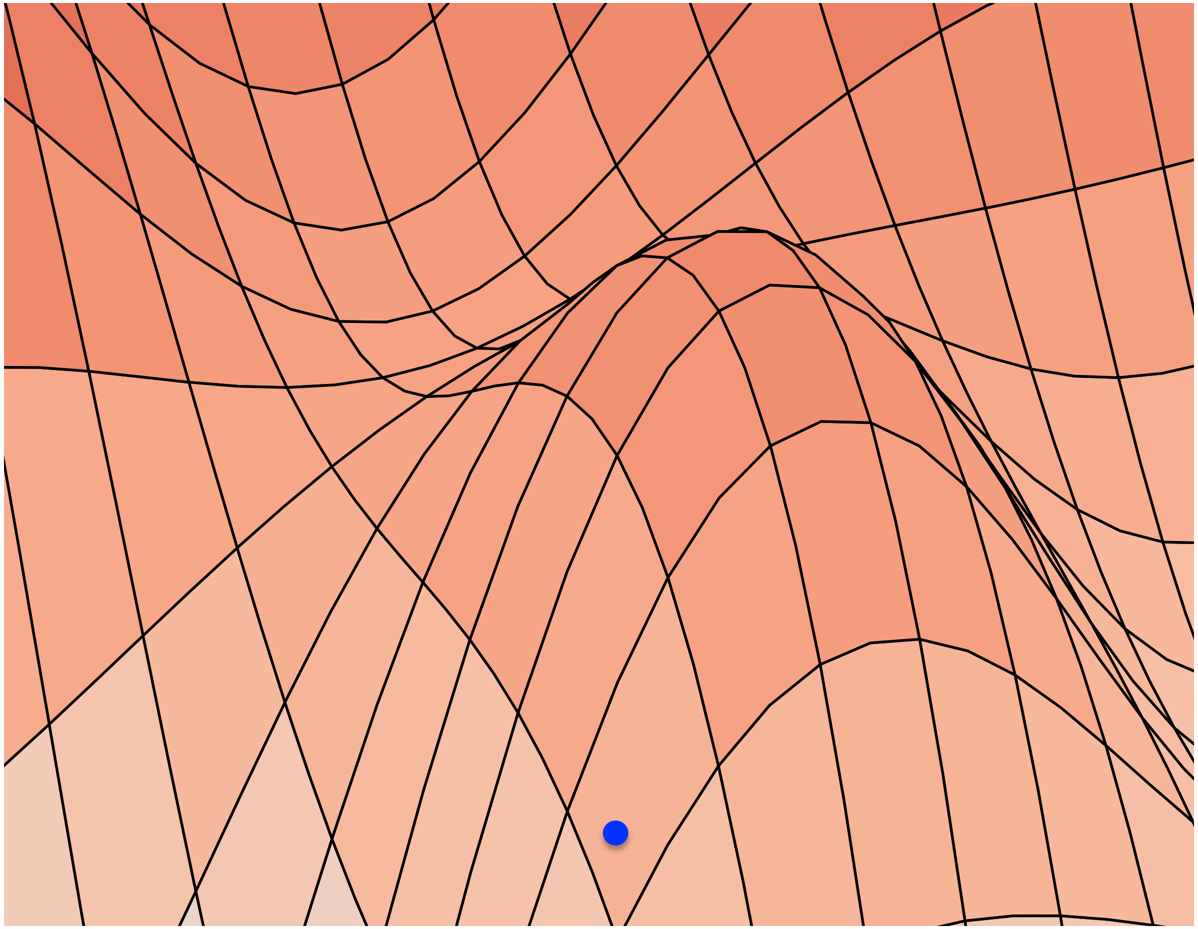


Training

Problem A: *Nonconvexity*

Stochastic Gradient Descent…

…climbs to the top of the nearest hill…

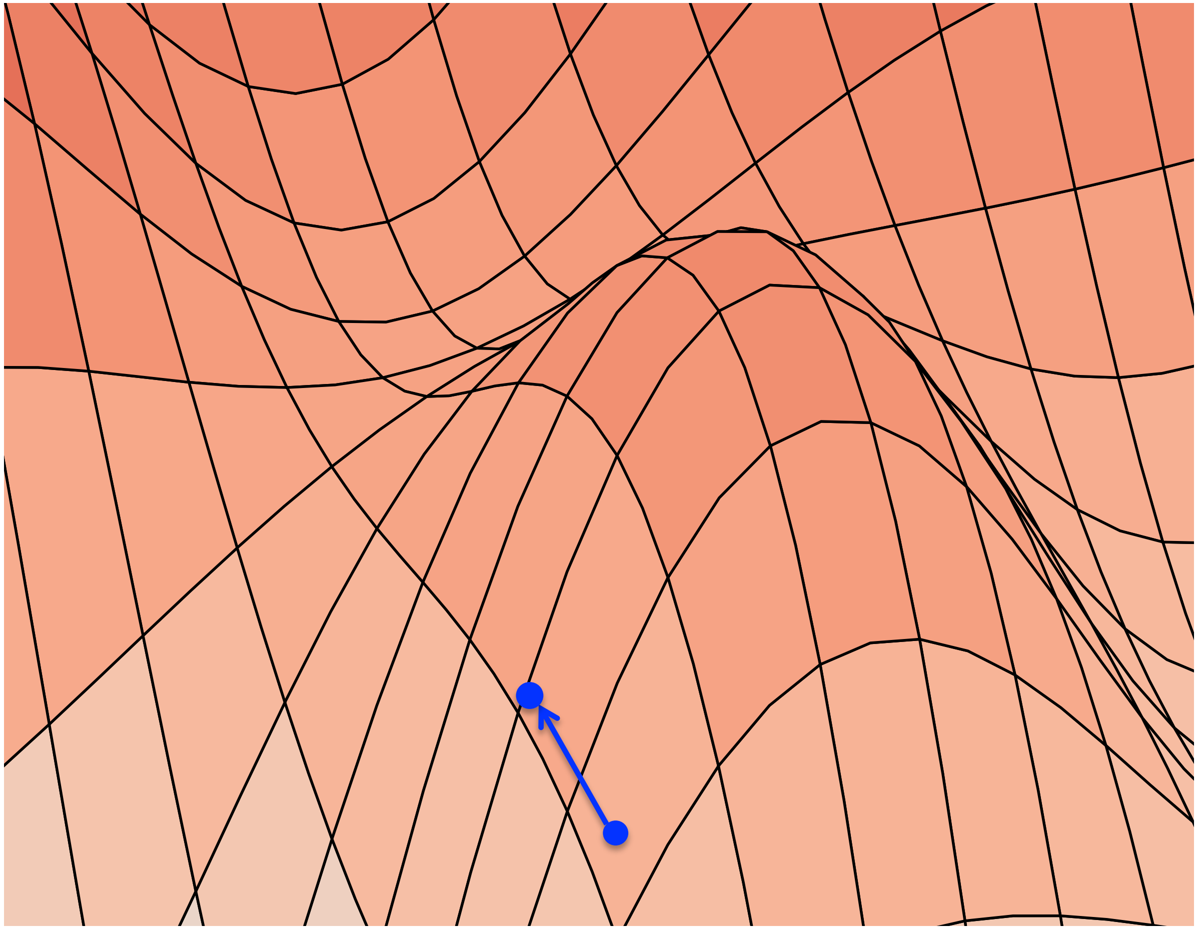


Training

Problem A: *Nonconvexity*

Stochastic Gradient Descent…

…climbs to the top of the nearest hill…

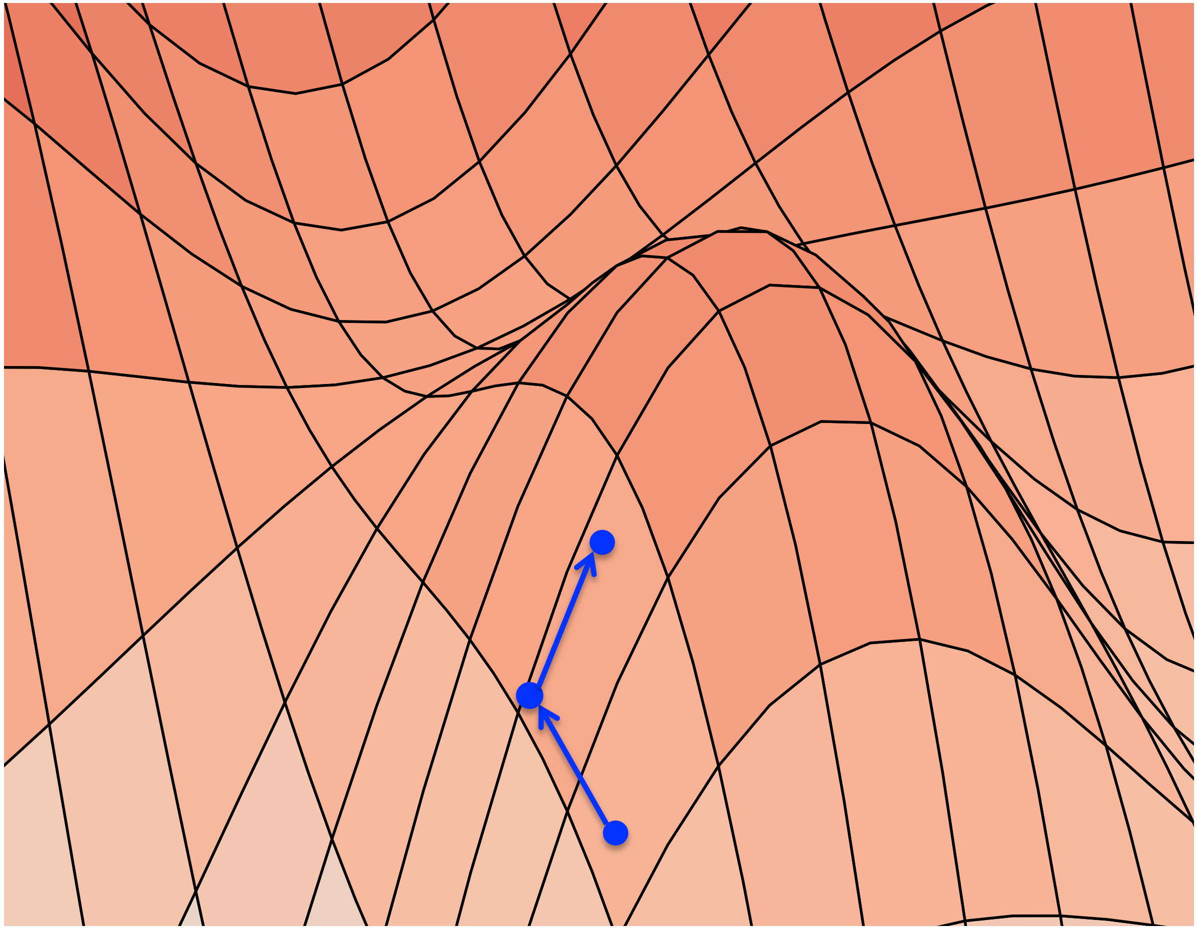


Training

Problem A: *Nonconvexity*

Stochastic Gradient Descent…

…climbs to the top of the nearest hill…

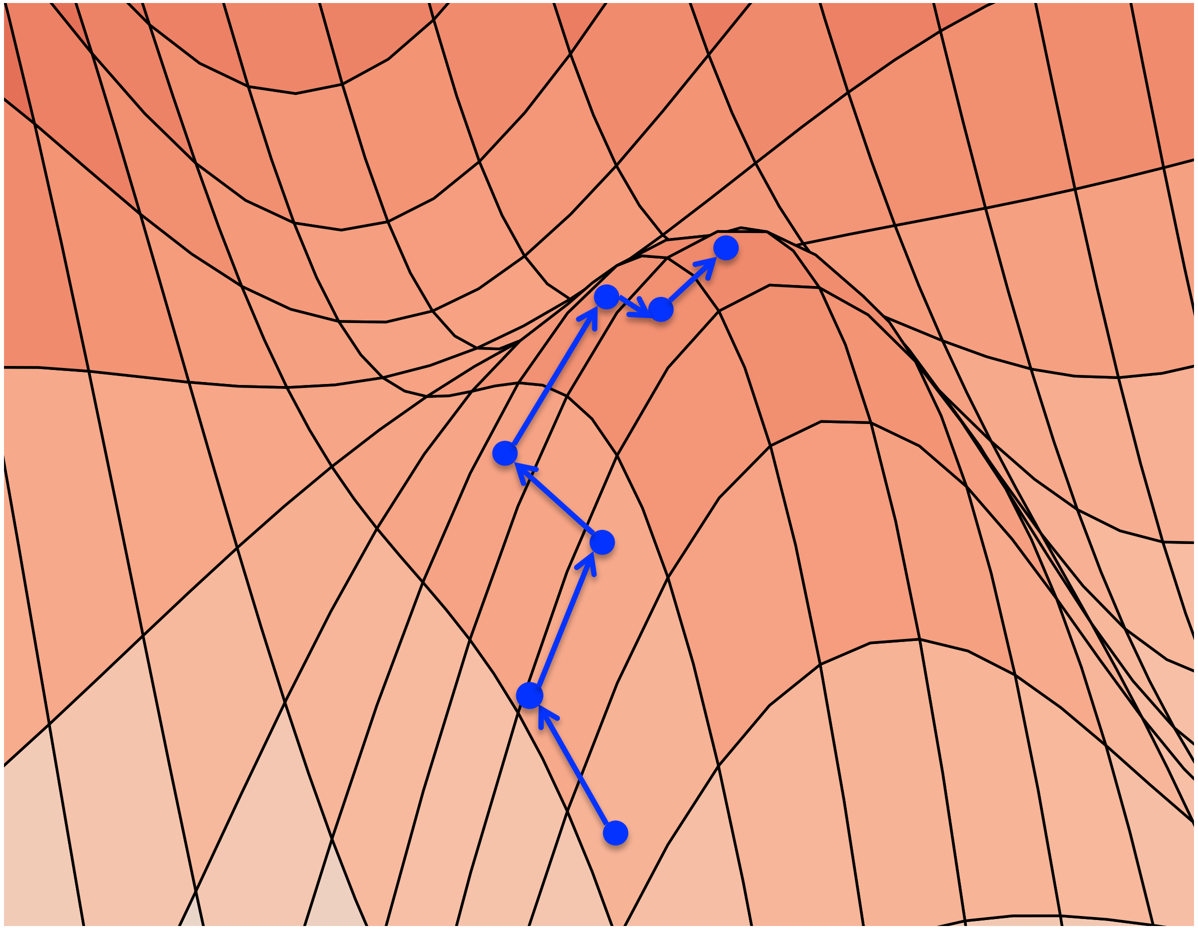


Training

Problem A: *Nonconvexity*

Stochastic Gradient Descent…

…climbs to the top of the nearest hill…



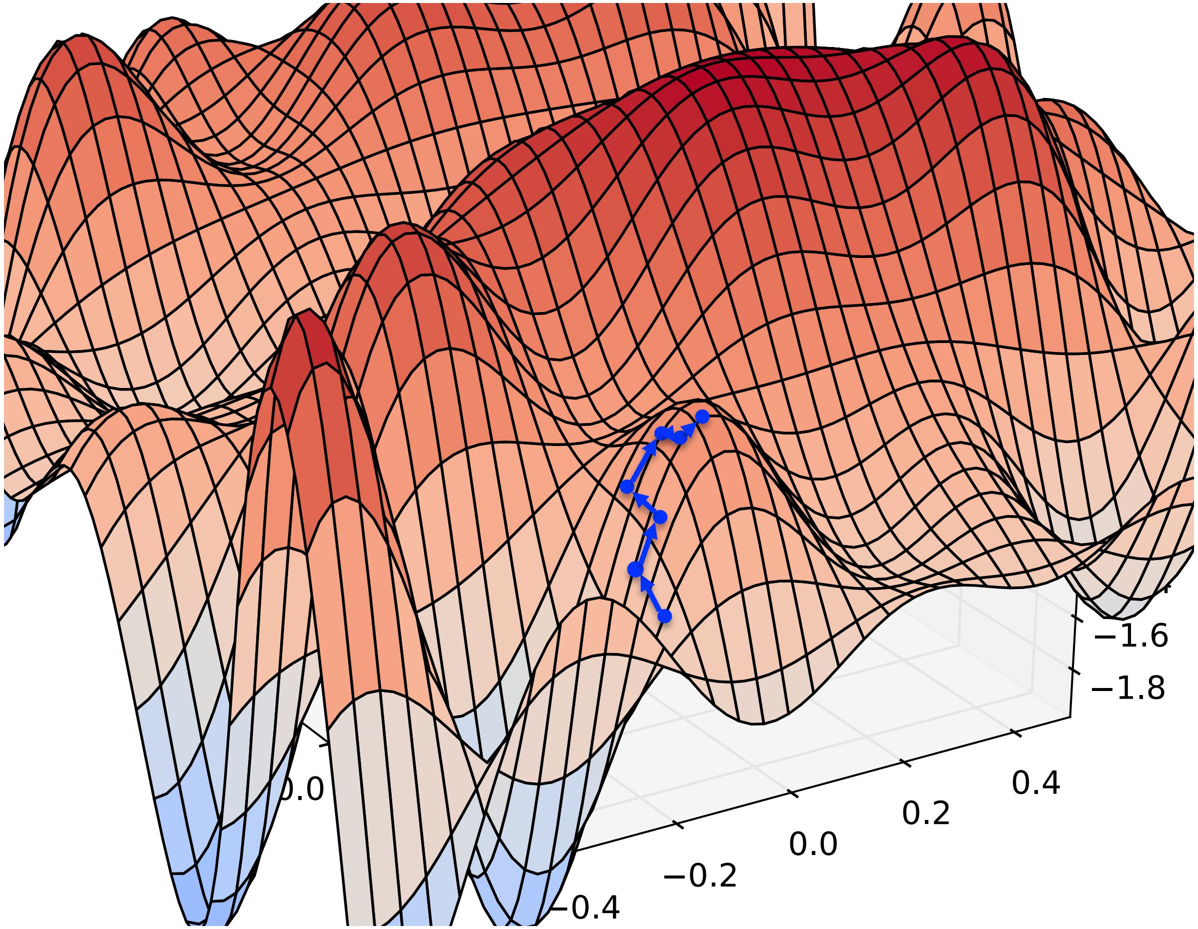
Training

Problem A: *Nonconvexity*

Stochastic Gradient Descent…

… climbs to the top of the nearest hill…

… which might not lead to the top of the mountain

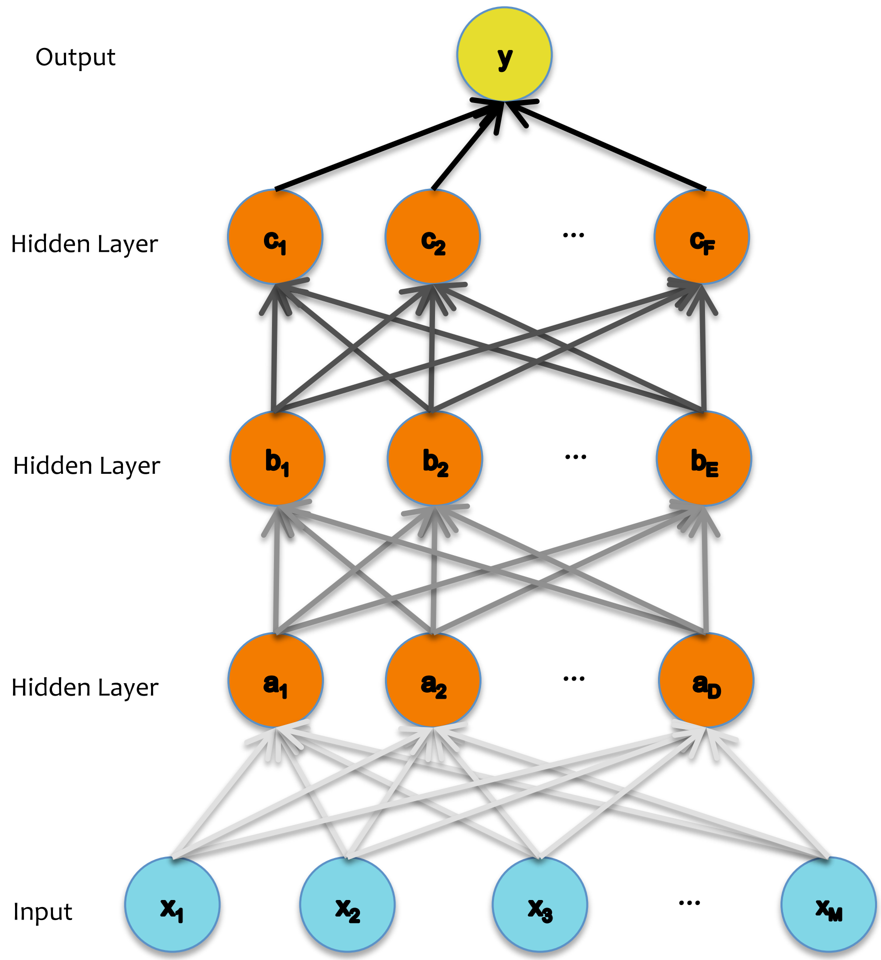


Training

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

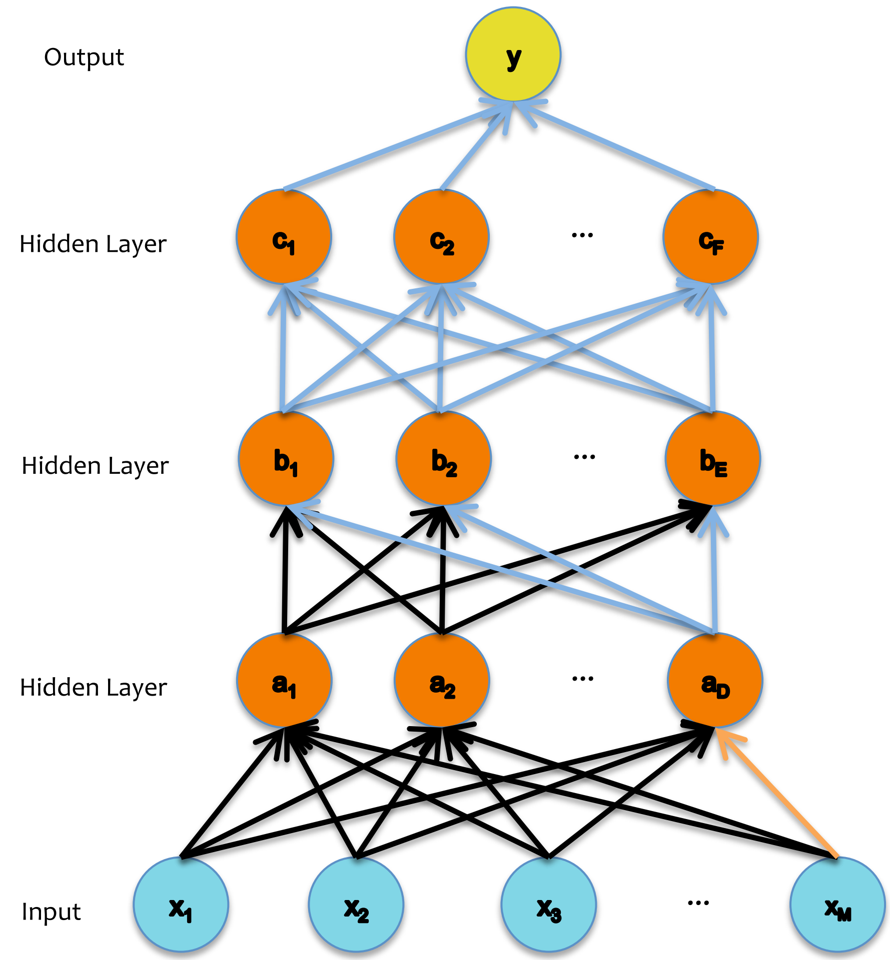


Training

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

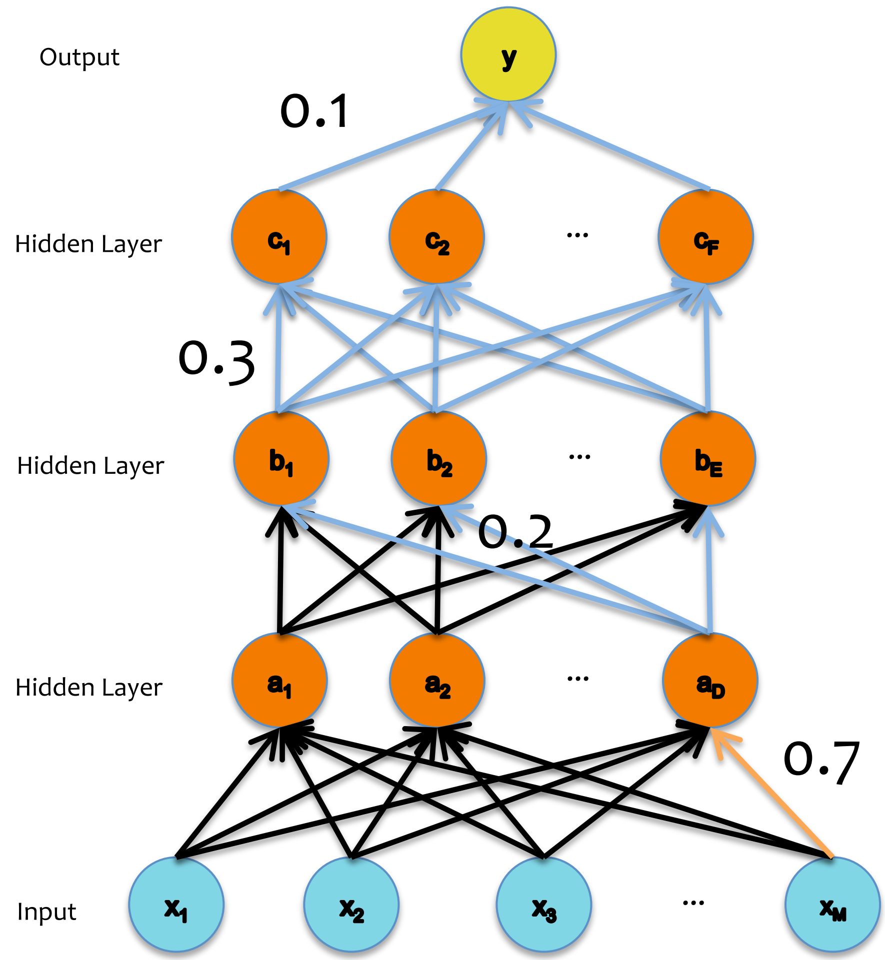


Training

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



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

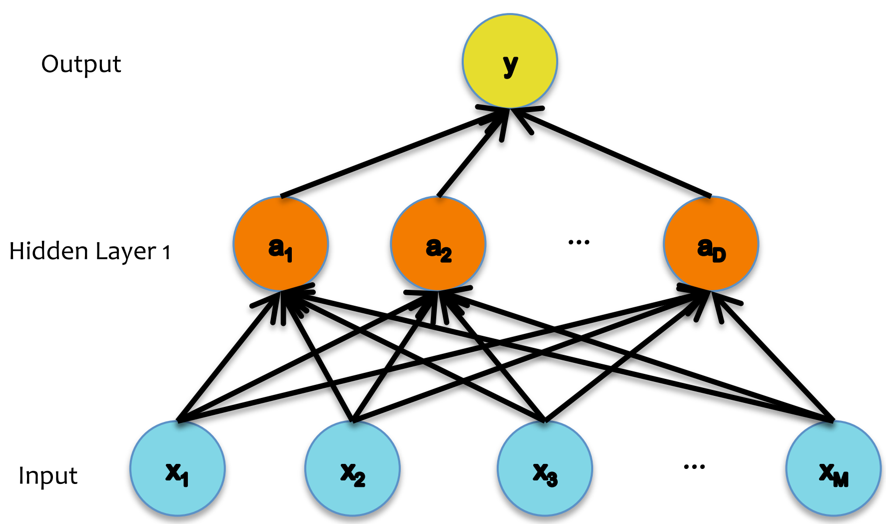
- **Idea #2: (Two Steps)**
  - **Train each level** of the model in a **greedy** way
  - Then use our **original idea**

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

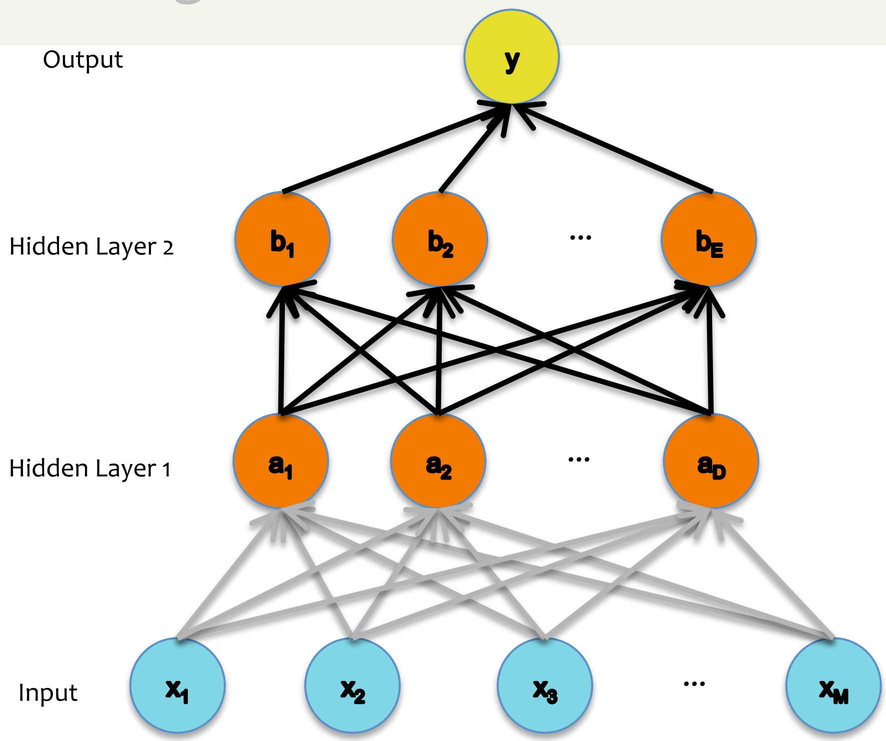
Idea #2: Supervised Pre-training

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



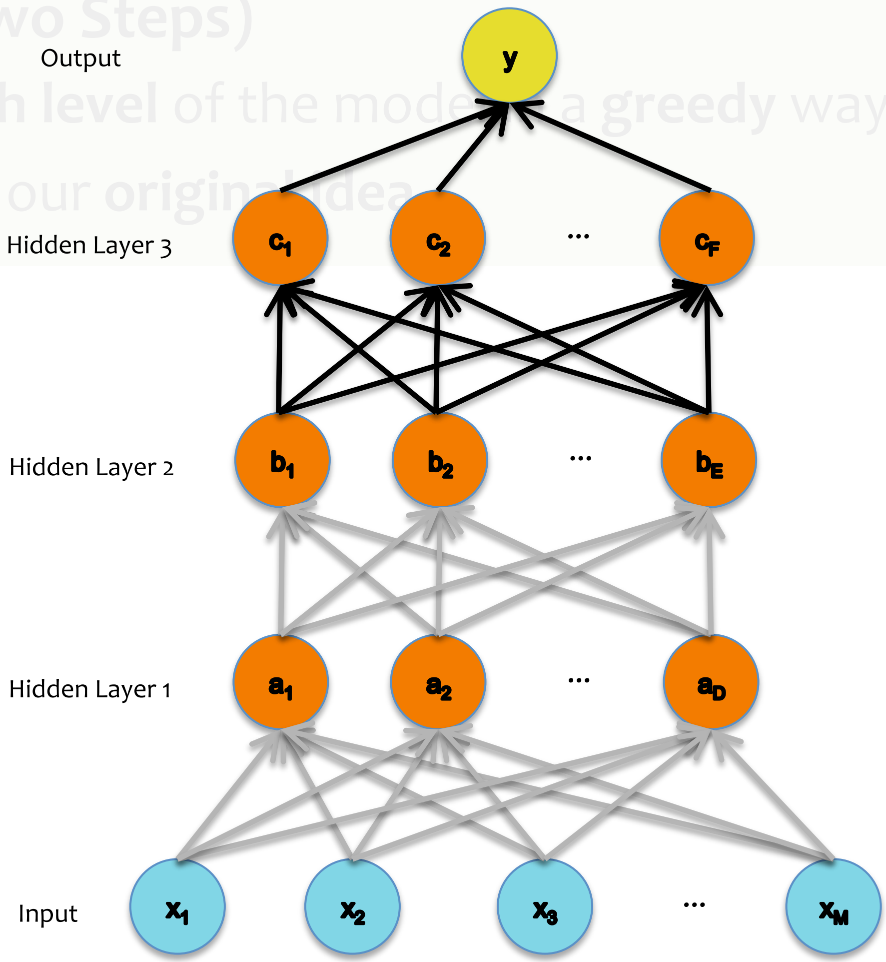
Idea #2: Supervised Pre-training

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



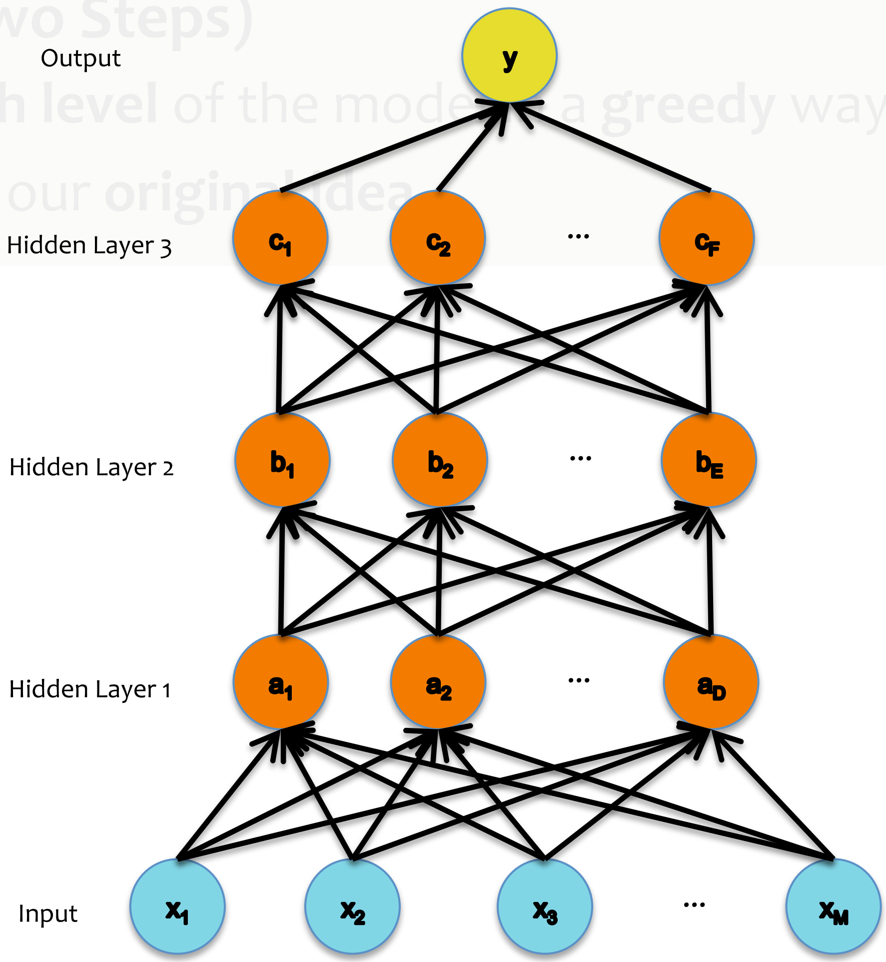
Idea #2: Supervised Pre-training

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



Idea #2: Supervised Pre-training

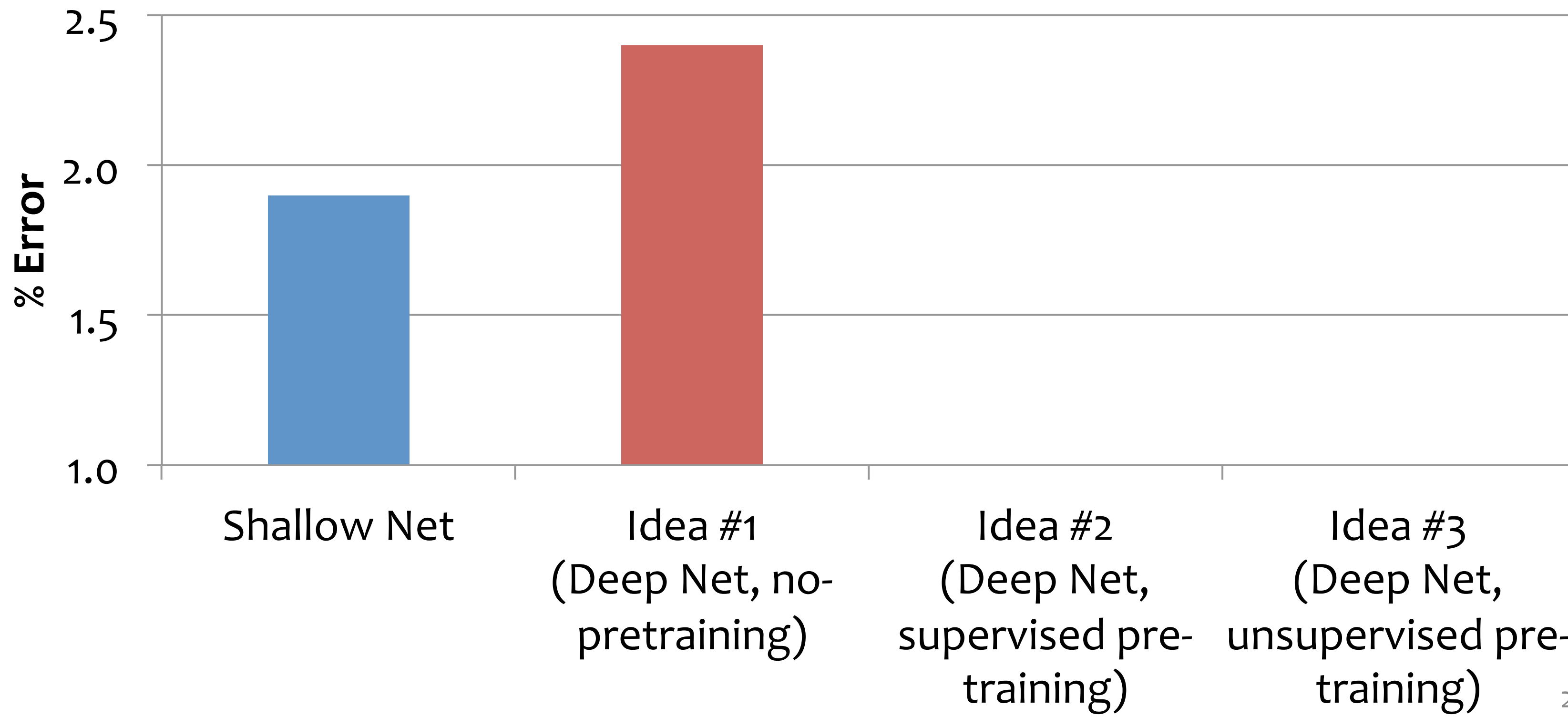
- Idea #2: (Two Steps)
  - Train each level of the mode a greedy way
  - Then use our origin idea



Training

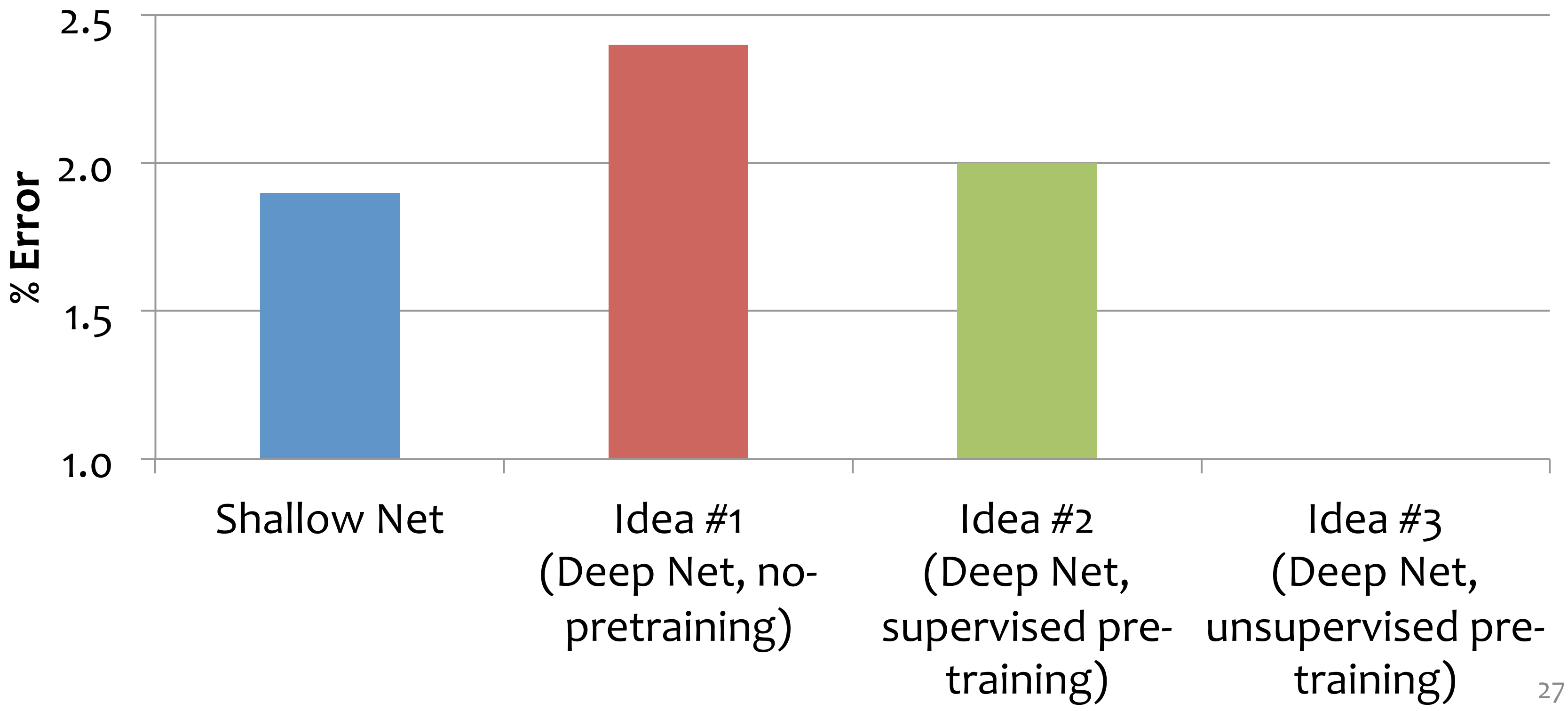
Comparison on MNIST

- 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 #3: Unsupervised Pre-training

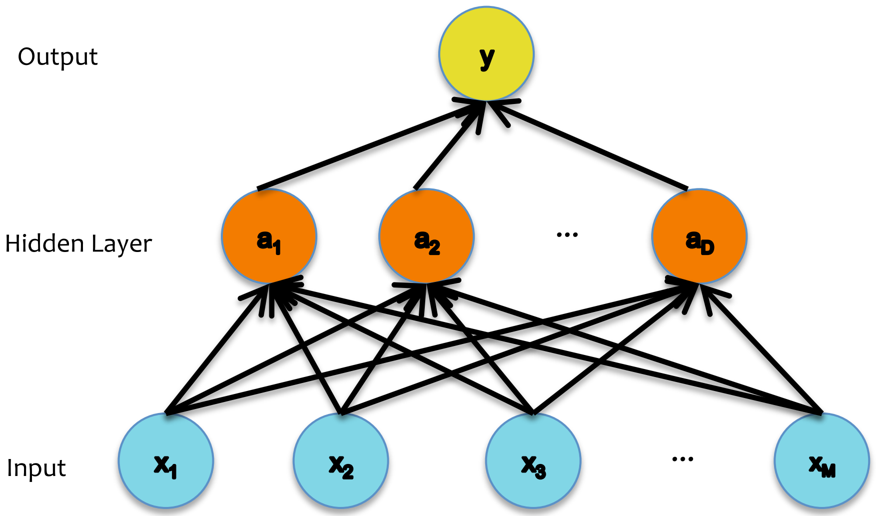
- **Idea #3: (Two Steps)**
  - Use our original idea, but **pick a better starting point**
  - **Train each level** of the model in a **greedy** way

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!**

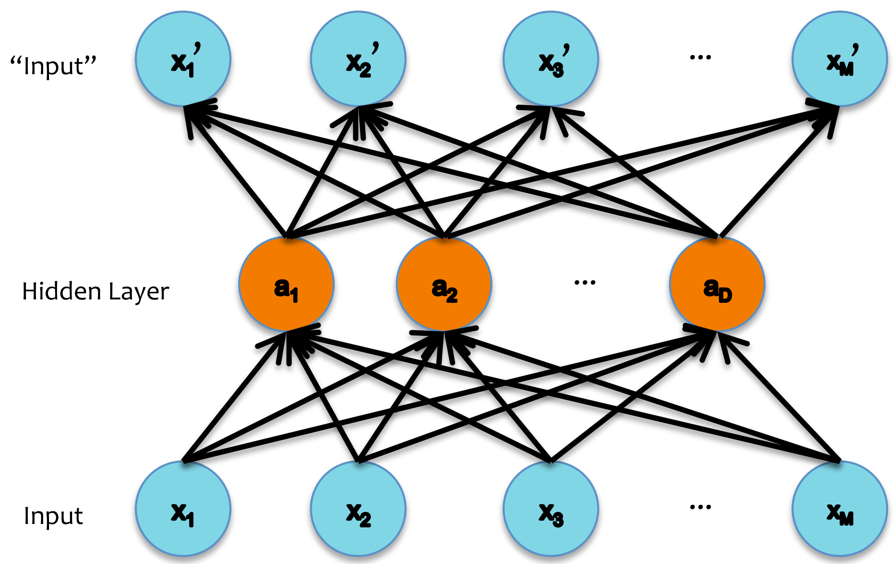


The solution: *Unsupervised pre-training*

Unsupervised pre-training of the first layer:

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

This topology defines an Auto-encoder.



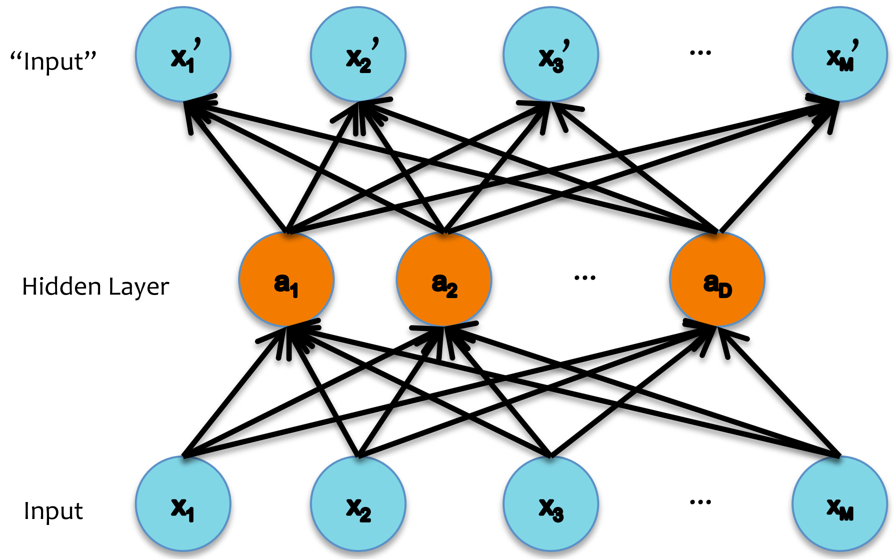
Auto-Encoders

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)

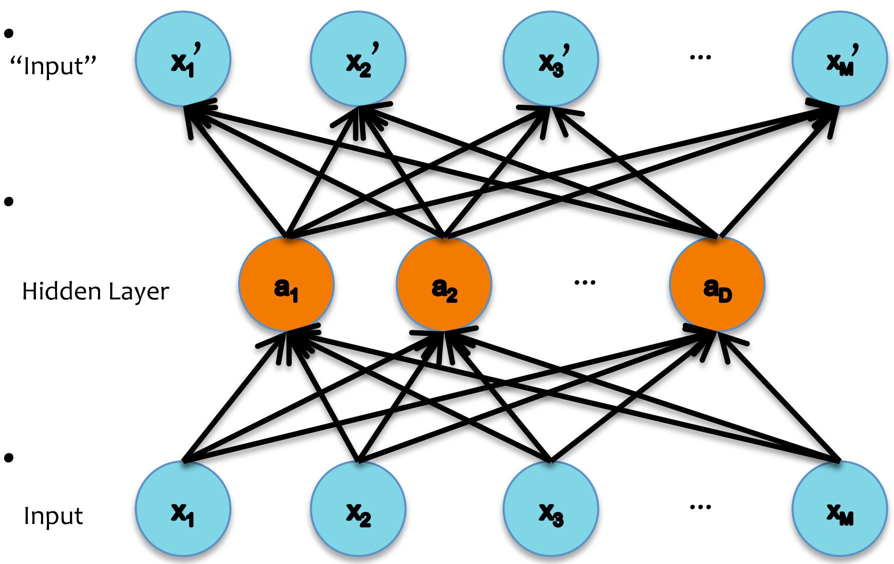


Slide adapted from Raman Arora

The solution:
Unsupervised pre-training

Unsupervised pre-training

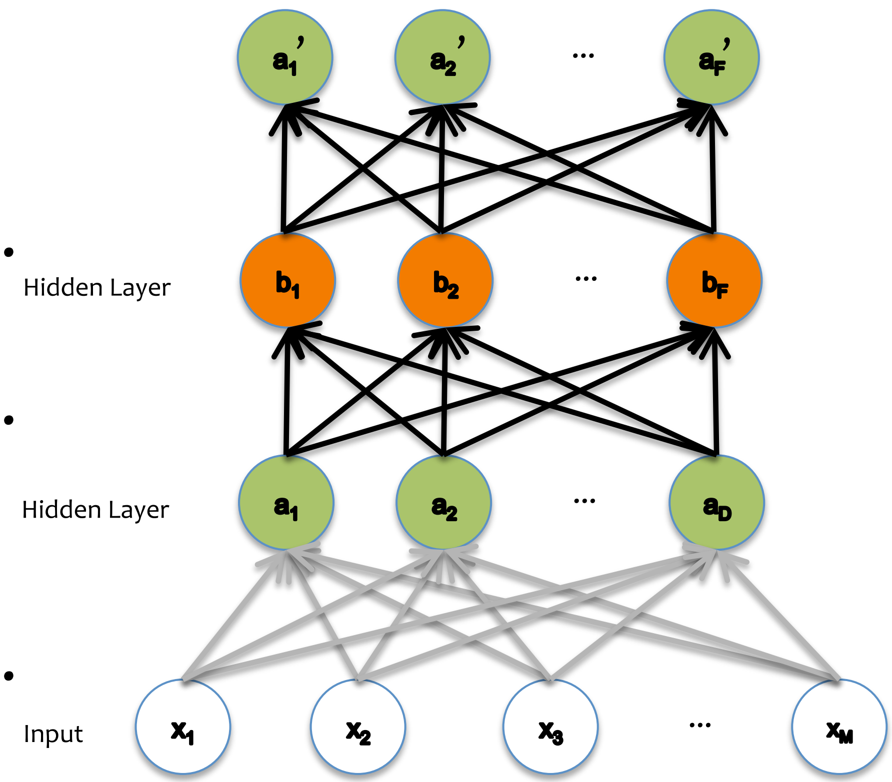
- Work bottom-up
  - Train hidden layer 1. Then fix its parameters.
  - Train hidden layer 2. Then fix its parameters.
  - …
  - Train hidden layer n. Then fix its parameters.



The solution:
Unsupervised pre-training

Unsupervised pre-training

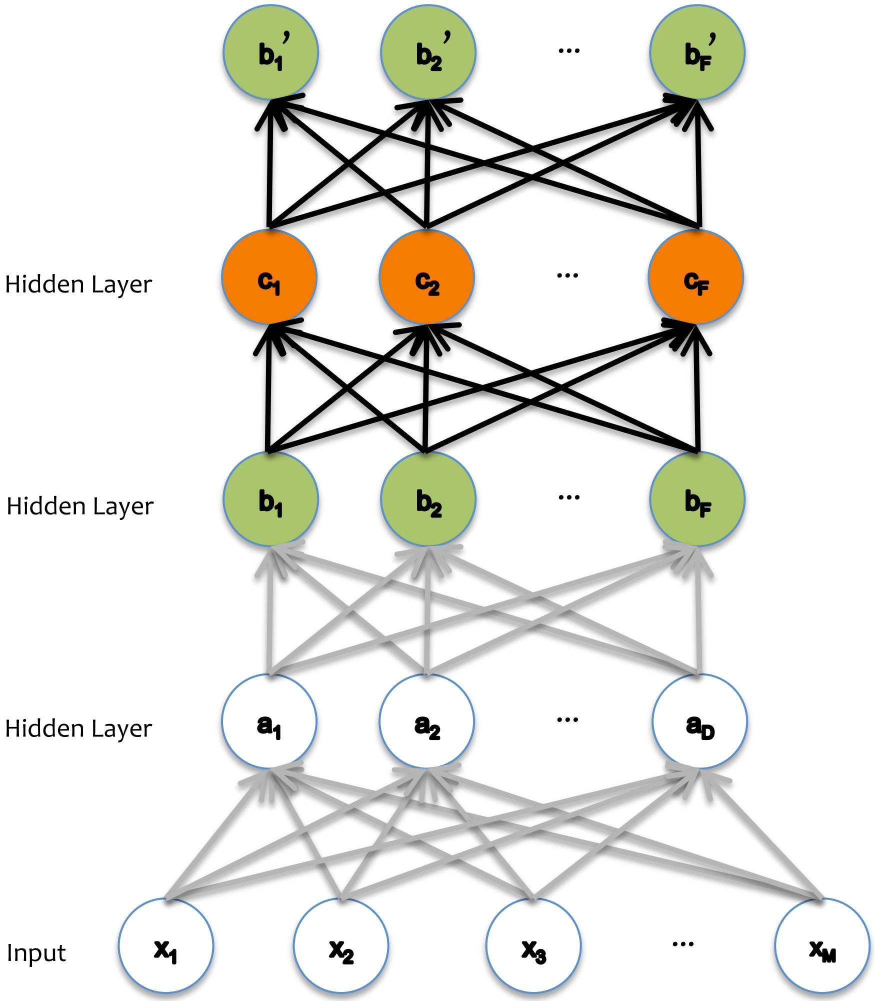
- Work bottom-up
  - Train hidden layer 1. Then fix its parameters.
  - Train hidden layer 2. Then fix its parameters.
  - …
  - Train hidden layer n. Then fix its parameters.



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.



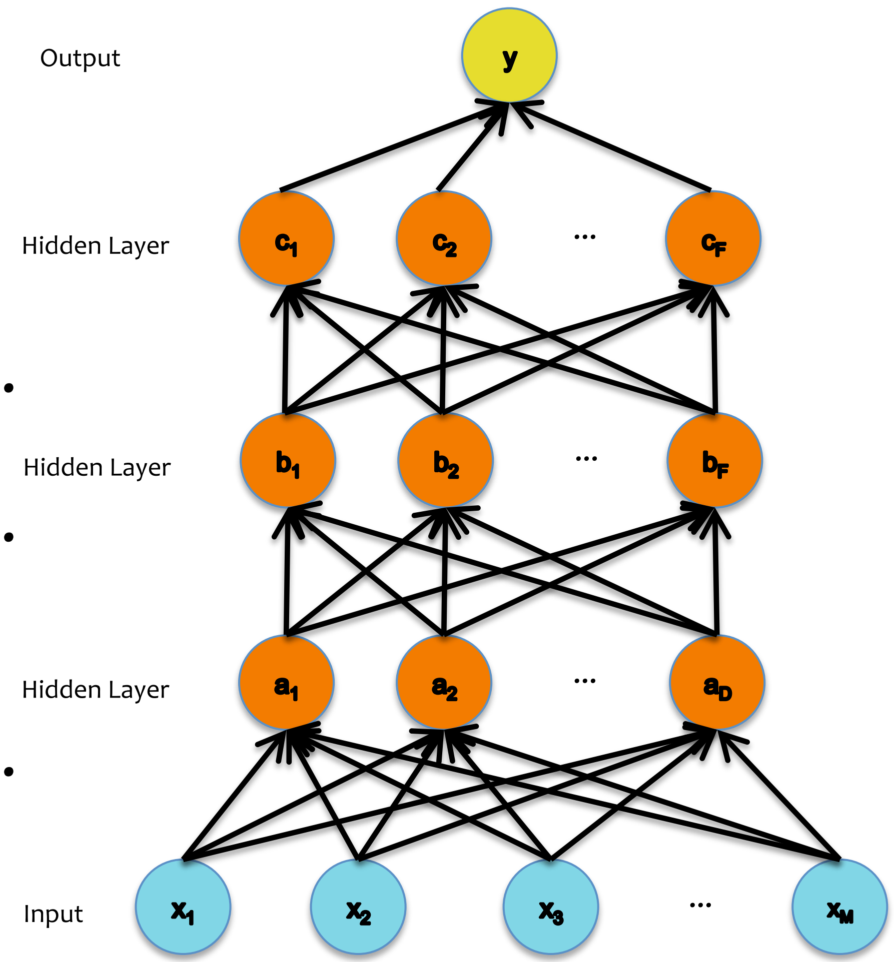
The solution: *Unsupervised pre-training*

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

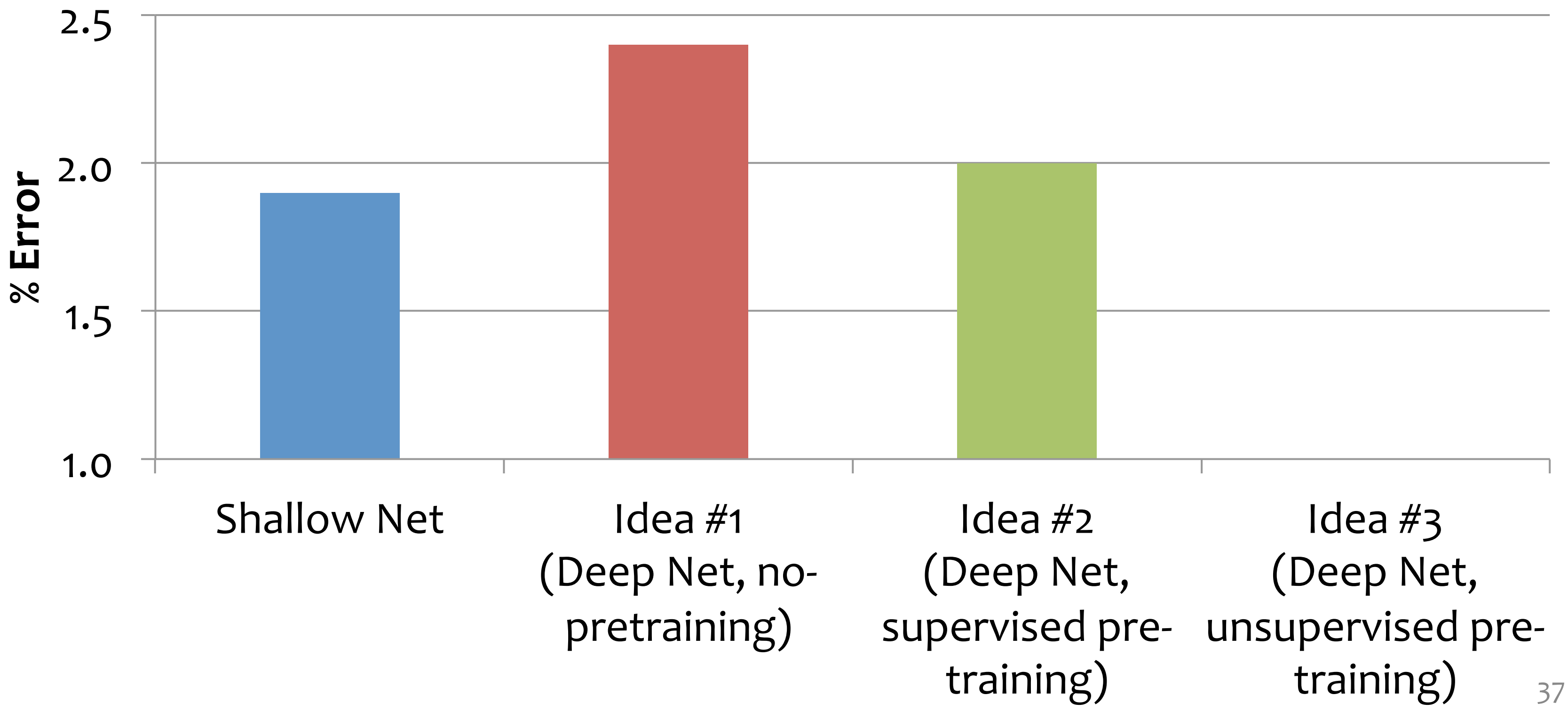
- **Idea #1:**
  1. Supervised fine-tuning only

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

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

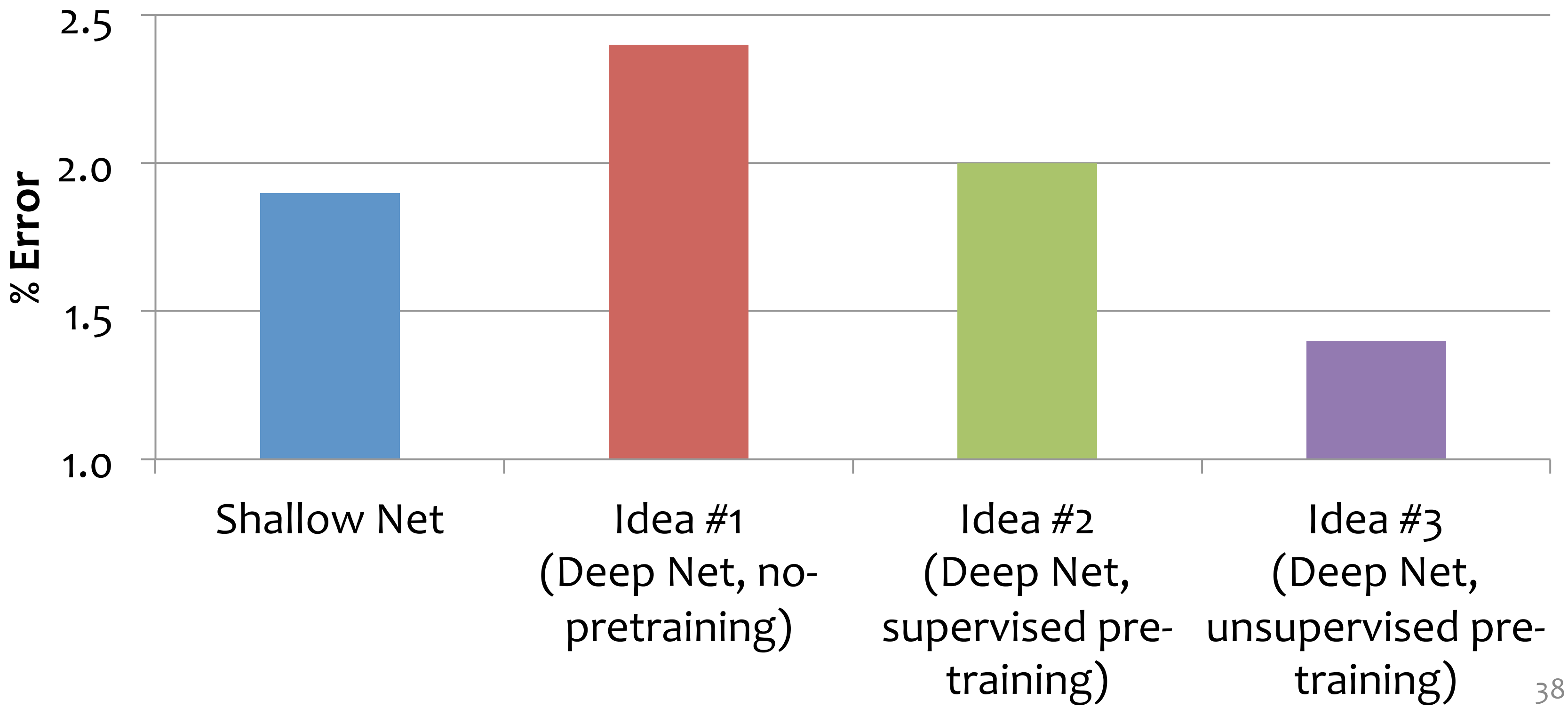
Comparison on MNIST

- 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)



Training

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!

RECURRENT NEURAL NETWORKS

Recurrent Neural Networks (RNNs)

inputs: $\mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I$

hidden units: $\mathbf{h} = (h_1, h_2, \ldots, h_T), h_i \in \mathcal{R}^J$

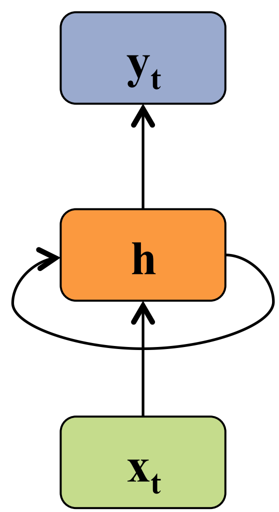
outputs: $\mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K$

nonlinearity: $\mathcal{H}$

Definition of the RNN:

$$h_t = \mathcal{H}\left(W_{xh}x_t + W_{hh}h_{t-1} + b_h\right)$$

$$y_t = W_{hy}h_t + b_y$$



Recurrent Neural Networks (RNNs)

inputs: $\mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I$

hidden units: $\mathbf{h} = (h_1, h_2, \ldots, h_T), h_i \in \mathcal{R}^J$

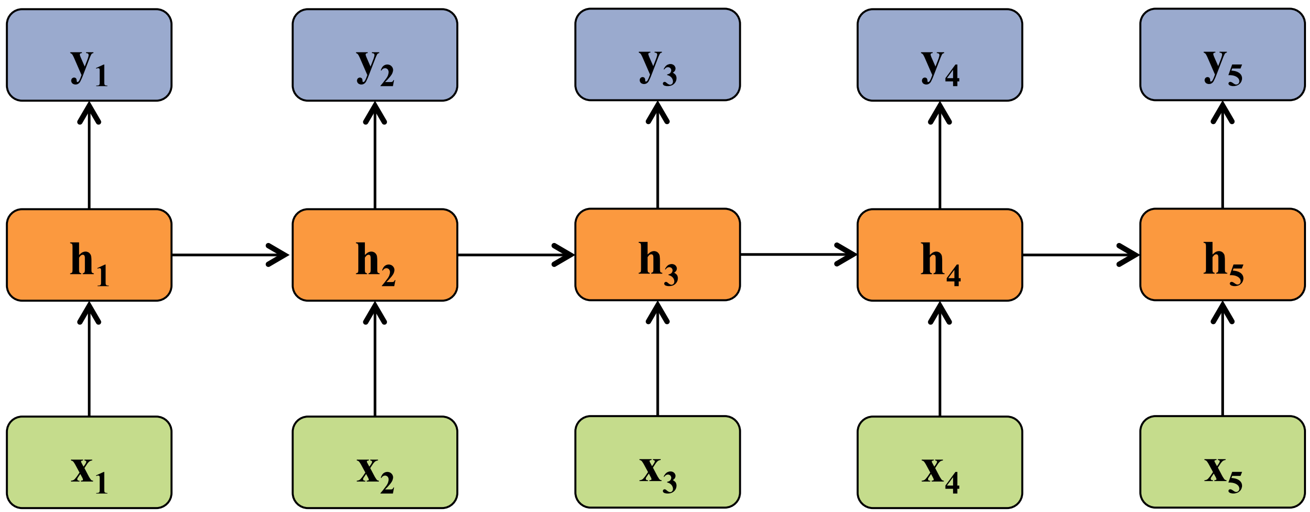
outputs: $\mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K$

nonlinearity: $\mathcal{H}$

Definition of the RNN:

$$h_t = \mathcal{H}\left(W_{xh} x_t + W_{hh} h_{t-1} + b_h\right)$$

$$y_t = W_{hy} h_t + b_y$$



Recurrent Neural Networks (RNNs)

inputs: $\mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I$

hidden units: $\mathbf{h} = (h_1, h_2, \ldots, h_T), h_i \in \mathcal{R}^J$

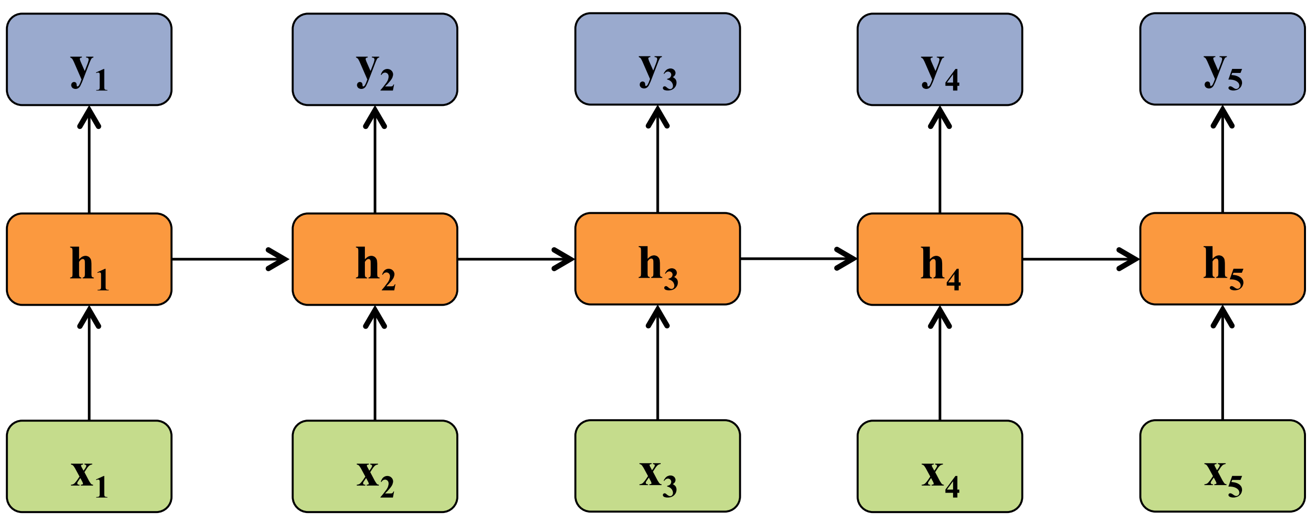
outputs: $\mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K$

nonlinearity: $\mathcal{H}$

Definition of the RNN:

$$h_t = \mathcal{H}\left(W_{xh}x_t + W_{hh}h_{t-1} + b_h\right)$$

$$y_t = W_{hy}h_t + b_y$$



Recurrent Neural Networks (RNNs)

inputs: $\mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I$

hidden units: $\mathbf{h} = (h_1, h_2, \ldots, h_T), h_i \in \mathcal{R}^J$

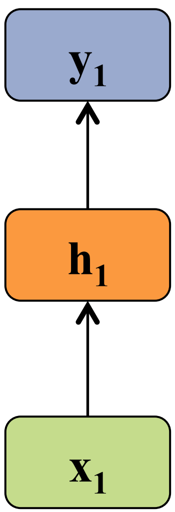
outputs: $\mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K$

nonlinearity: $\mathcal{H}$

Definition of the RNN:

$$h_t = \mathcal{H}\left(W_{xh} x_t + W_{hh} h_{t-1} + b_h\right)$$

$$y_t = W_{hy} h_t + b_y$$



- If $T=1$, then we have a standard feed-forward **neural net with one hidden layer**
- All of the deep nets from last lecture (DNN, DBN, DBM) required **fixed size inputs/outputs**

Recurrent Neural Networks (RNNs)

inputs: $\mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I$

hidden units: $\mathbf{h} = (h_1, h_2, \ldots, h_T), h_i \in \mathcal{R}^J$

outputs: $\mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K$

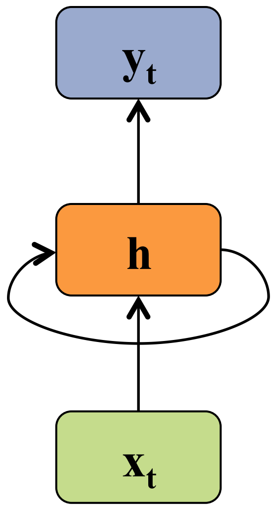
nonlinearity: $\mathcal{H}$

Definition of the RNN:

$$h_t = \mathcal{H}\left(W_{xh} x_t + W_{hh} h_{t-1} + b_h\right)$$

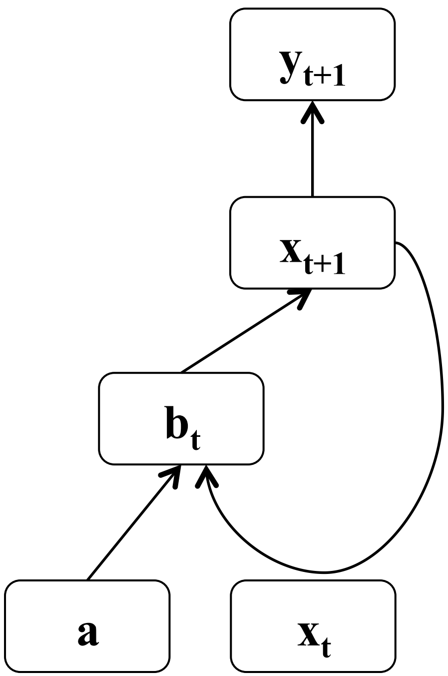
$$y_t = W_{hy} h_t + b_y$$

- By unrolling the RNN through time, we can **share parameters** and accommodate **arbitrary length** input/output pairs
- Applications: **time-series data** such as sentences, speech, stock-market, signal data, etc.



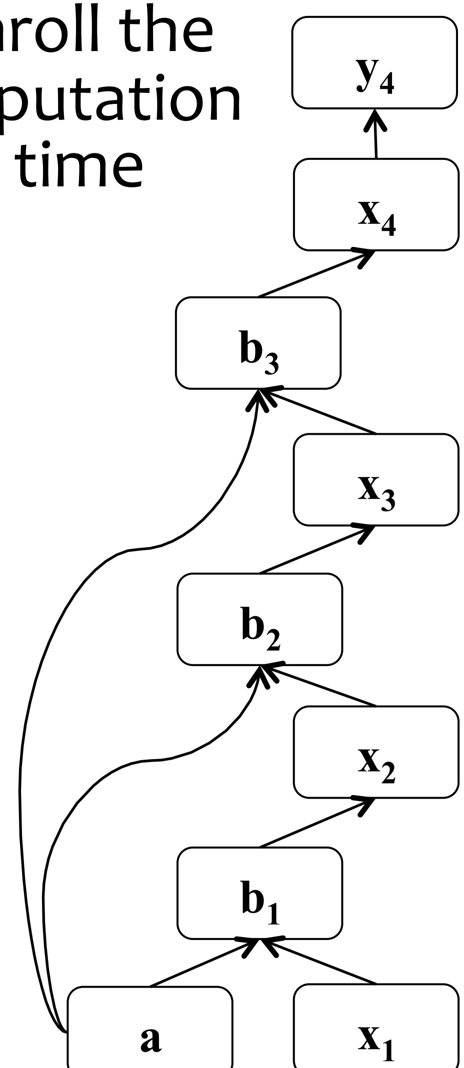
Background: Backprop through time

Recurrent neural network:



BPTT:

1. Unroll the computation over time



2. Run backprop through the resulting feed-forward network

(Robinson & Fallside, 1987)
(Werbos, 1988)
(Mozer, 1995)

Bidirectional RNN

$$\text{inputs: } \mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I$$

$$\text{hidden units: } \overrightarrow{\mathbf{h}} \text{ and } \overleftarrow{\mathbf{h}}$$

$$\text{outputs: } \mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K$$

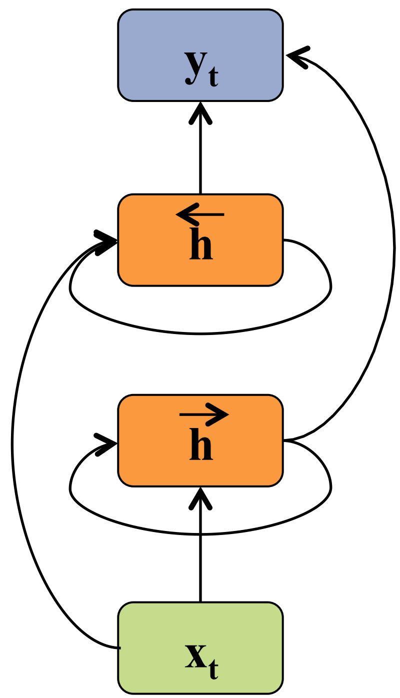
$$\text{nonlinearity: } \mathcal{H}$$

Recursive Definition:

$$\overrightarrow{h}_t = \mathcal{H}\left(W_{x\overrightarrow{h}} x_t + W_{\overrightarrow{h}\overrightarrow{h}} \overrightarrow{h}_{t-1} + b_{\overrightarrow{h}}\right)$$

$$\overleftarrow{h}_t = \mathcal{H}\left(W_{x\overleftarrow{h}} x_t + W_{\overleftarrow{h}\overleftarrow{h}} \overleftarrow{h}_{t+1} + b_{\overleftarrow{h}}\right)$$

$$y_t = W_{\overrightarrow{h}y} \overrightarrow{h}_t + W_{\overleftarrow{h}y} \overleftarrow{h}_t + b_y$$



Bidirectional RNN

inputs: $\mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I$

hidden units: $\overrightarrow{\mathbf{h}}$ and $\overleftarrow{\mathbf{h}}$

outputs: $\mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K$

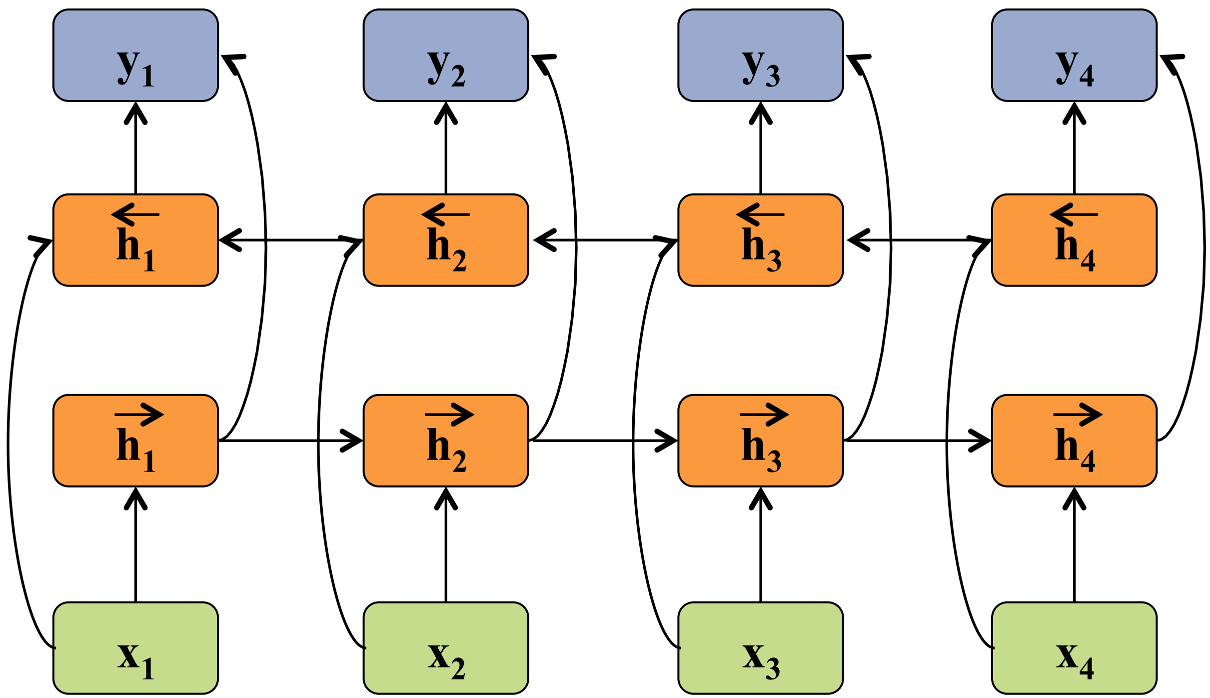
nonlinearity: $\mathcal{H}$

Recursive Definition:

$$\overrightarrow{h}_t = \mathcal{H}\left(W_{x\overrightarrow{h}} x_t + W_{\overrightarrow{h}\overrightarrow{h}} \overrightarrow{h}_{t-1} + b_{\overrightarrow{h}}\right)$$

$$\overleftarrow{h}_t = \mathcal{H}\left(W_{x\overleftarrow{h}} x_t + W_{\overleftarrow{h}\overleftarrow{h}} \overleftarrow{h}_{t+1} + b_{\overleftarrow{h}}\right)$$

$$y_t = W_{\overrightarrow{h}y} \overrightarrow{h}_t + W_{\overleftarrow{h}y} \overleftarrow{h}_t + b_y$$



Bidirectional RNN

inputs: $\mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I$

hidden units: $\overrightarrow{\mathbf{h}}$ and $\overleftarrow{\mathbf{h}}$

outputs: $\mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K$

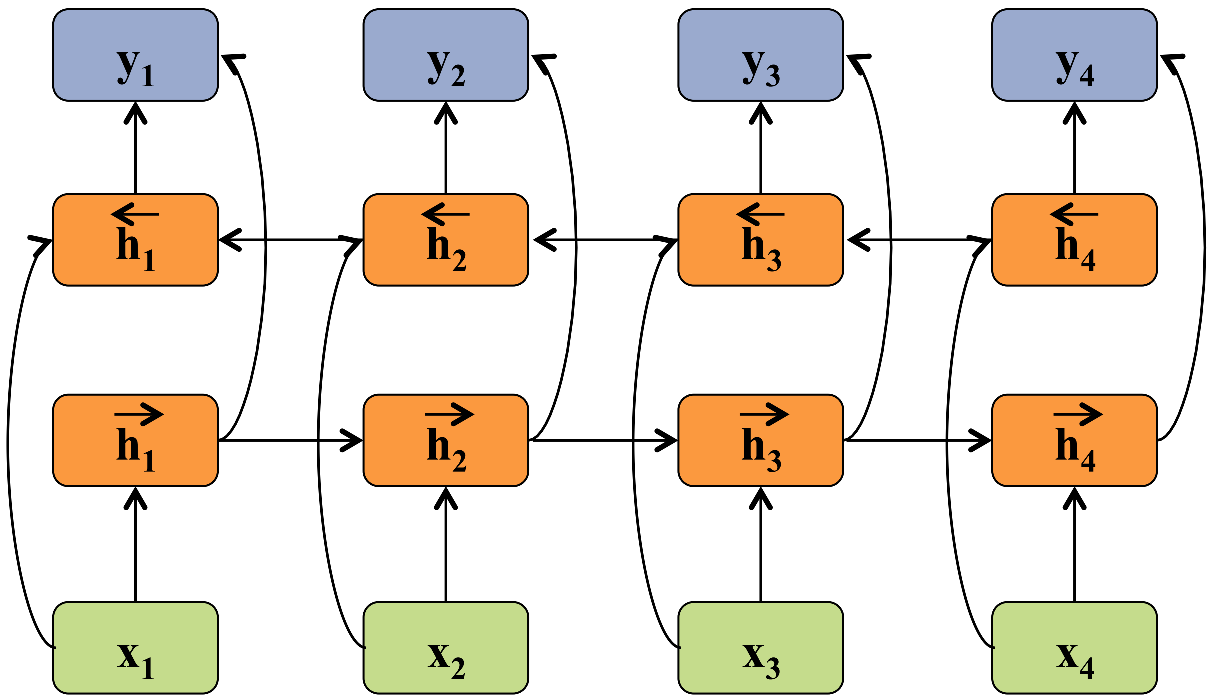
nonlinearity: $\mathcal{H}$

Recursive Definition:

$$\overrightarrow{h}_t = \mathcal{H}\left(W_{x\overrightarrow{h}} x_t + W_{\overrightarrow{h}\overrightarrow{h}} \overrightarrow{h}_{t-1} + b_{\overrightarrow{h}}\right)$$

$$\overleftarrow{h}_t = \mathcal{H}\left(W_{x\overleftarrow{h}} x_t + W_{\overleftarrow{h}\overleftarrow{h}} \overleftarrow{h}_{t+1} + b_{\overleftarrow{h}}\right)$$

$$y_t = W_{\overrightarrow{h}y} \overrightarrow{h}_t + W_{\overleftarrow{h}y} \overleftarrow{h}_t + b_y$$



Bidirectional RNN

inputs: $\mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I$

hidden units: $\overrightarrow{\mathbf{h}}$ and $\overleftarrow{\mathbf{h}}$

outputs: $\mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K$

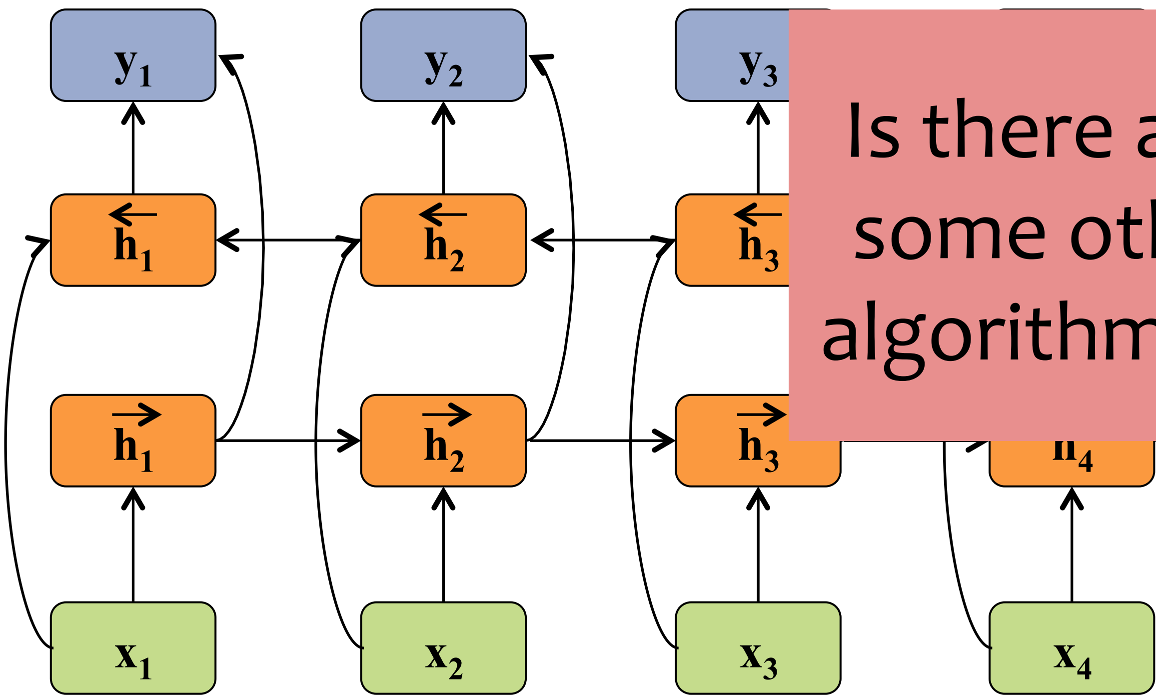
nonlinearity: $\mathcal{H}$

Recursive Definition:

$$\overrightarrow{h}_t = \mathcal{H}\left(W_{x\overrightarrow{h}} x_t + W_{\overrightarrow{h}\overrightarrow{h}} \overrightarrow{h}_{t-1} + b_{\overrightarrow{h}}\right)$$

$$\overleftarrow{h}_t = \mathcal{H}\left(W_{x\overleftarrow{h}} x_t + W_{\overleftarrow{h}\overleftarrow{h}} \overleftarrow{h}_{t+1} + b_{\overleftarrow{h}}\right)$$

$$y_t = W_{\overrightarrow{h}y} \overrightarrow{h}_t + W_{\overleftarrow{h}y} \overleftarrow{h}_t + b_y$$



Is there an analogy to some other recursive algorithm(s) we know?

Deep RNNs

inputs: $\mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I$

outputs: $\mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K$

nonlinearity: $\mathcal{H}$

Recursive Definition:

$$h_t^n = \mathcal{H}\left(W_{h^{n-1}h^n} h_t^{n-1} + W_{h^n h^n} h_{t-1}^n + b_h^n\right)$$

$$y_t = W_{h^N y} h_t^N + b_y$$

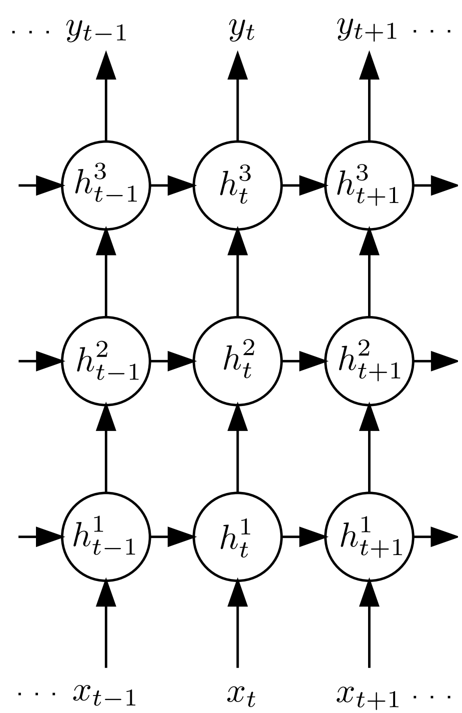


Figure from (Graves et al., 2013)

Deep Bidirectional RNNs

inputs: $\mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I$

outputs: $\mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K$

nonlinearity: $\mathcal{H}$

- Notice that the upper level hidden units have input from **two previous layers** (i.e. wider input)
- Likewise for the output layer
- What analogy can we draw to DNNs, DBNs, DBMs?

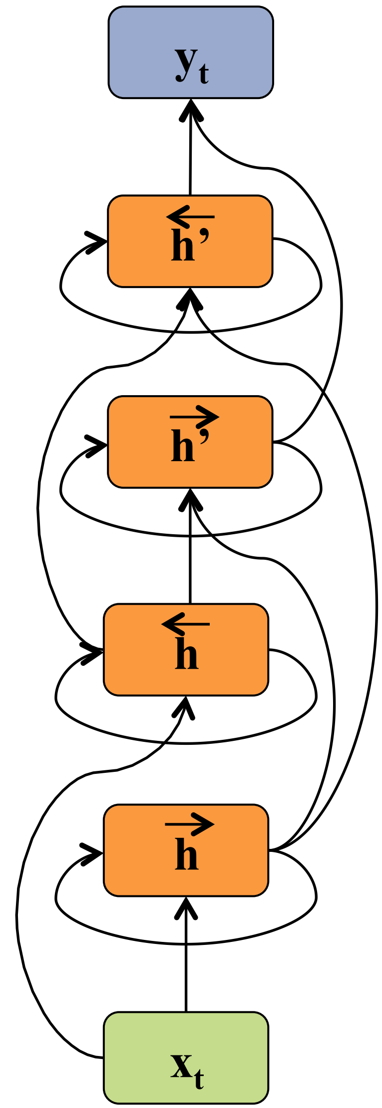
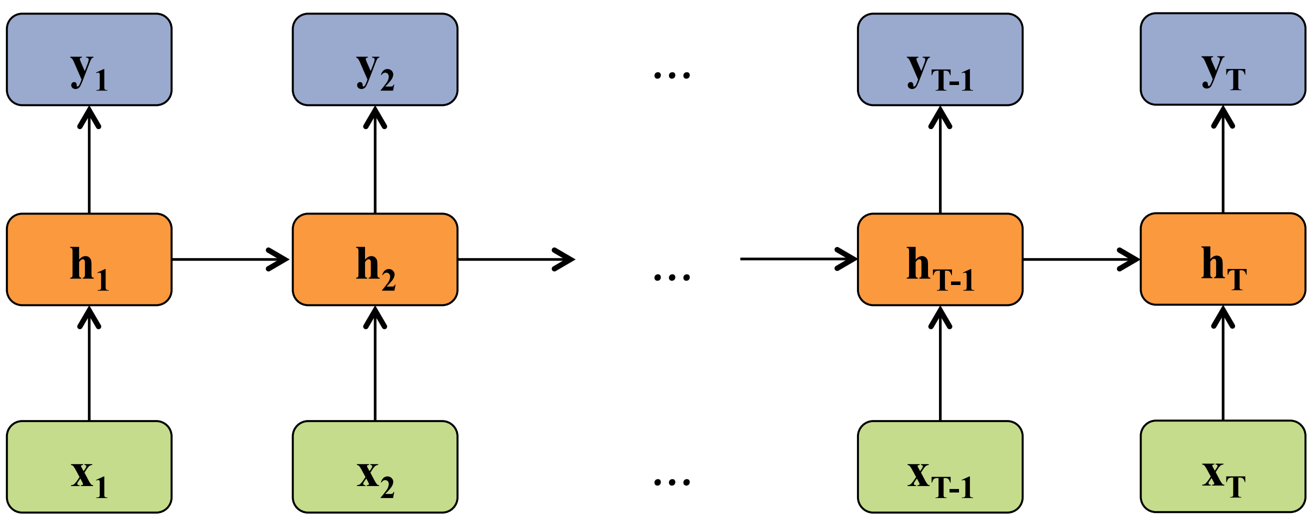


Figure from (Graves et al., 2013)

Long Short-Term Memory (LSTM)

Motivation:

- Standard RNNs have trouble learning long distance dependencies
- LSTMs combat this issue



Long Short-Term Memory (LSTM)

Motivation:

- Vanishing gradient problem for Standard RNNs
- Figure shows sensitivity (darker = more sensitive) to the input at time t=1

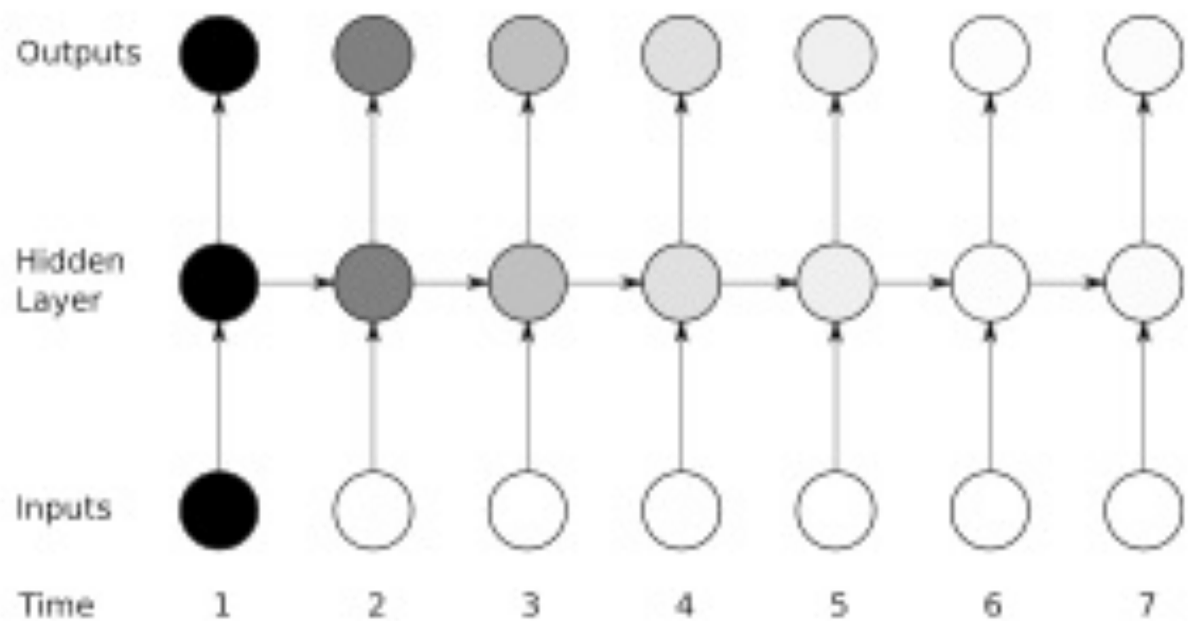


Figure from (Graves, 2012)

Long Short-Term Memory (LSTM)

Motivation:

- LSTM units have a rich internal structure
- The various "gates" determine the propagation of information and can choose to "remember" or "forget" information

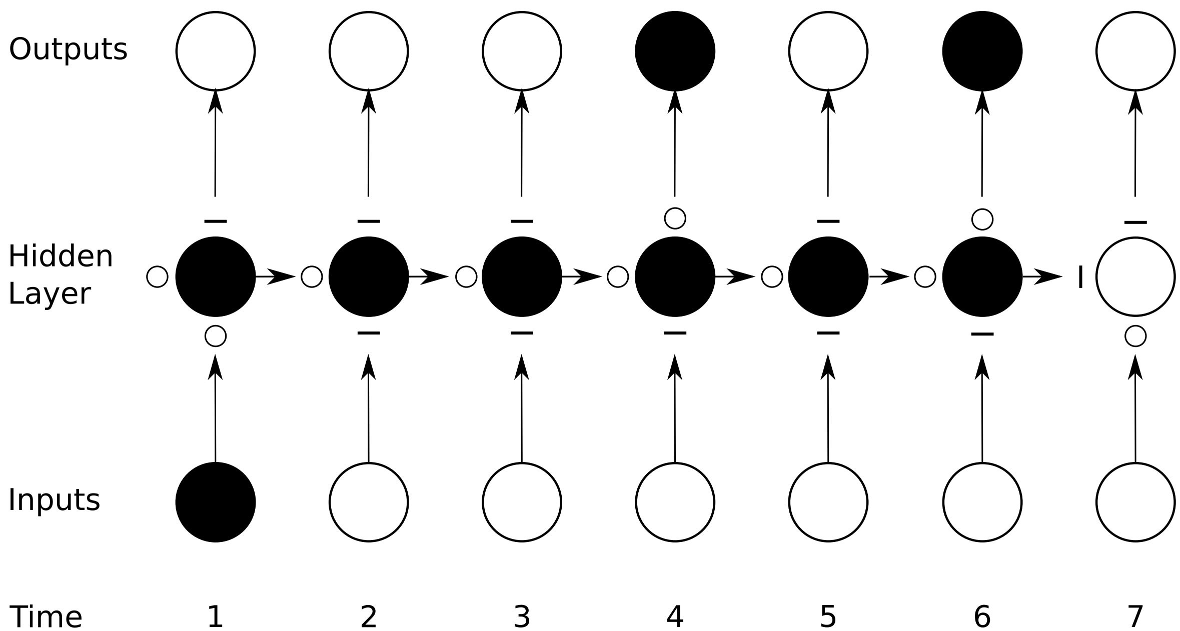
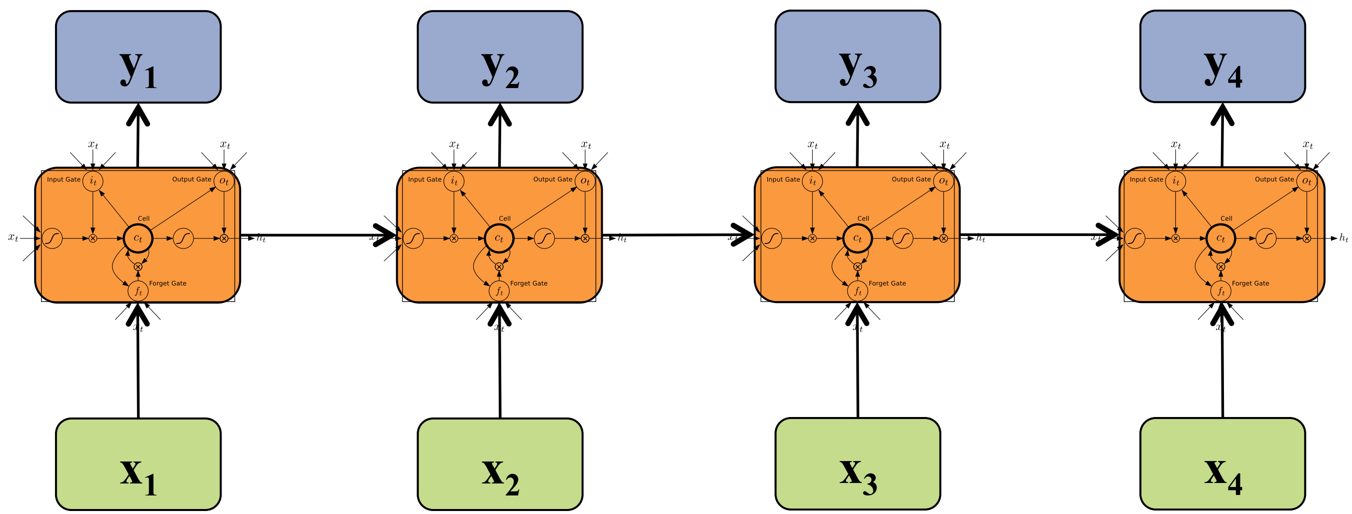


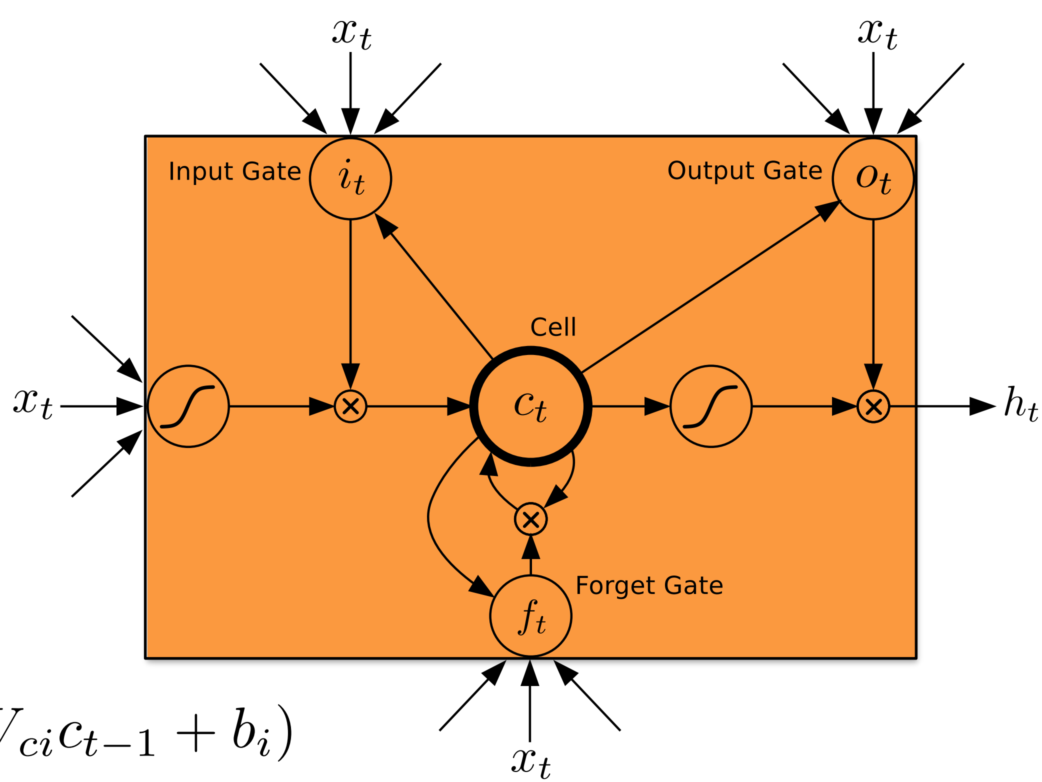
Figure from (Graves, 2012)

Long Short-Term Memory (LSTM)



Long Short-Term Memory (LSTM)

- **Input gate:** masks out the standard RNN inputs
- **Forget gate:** masks out the previous cell
- **Cell:** stores the input/forget mixture
- **Output gate:** masks out the values of the next hidden



$$i_t = \sigma\left(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i\right)$$

$$f_t = \sigma\left(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f\right)$$

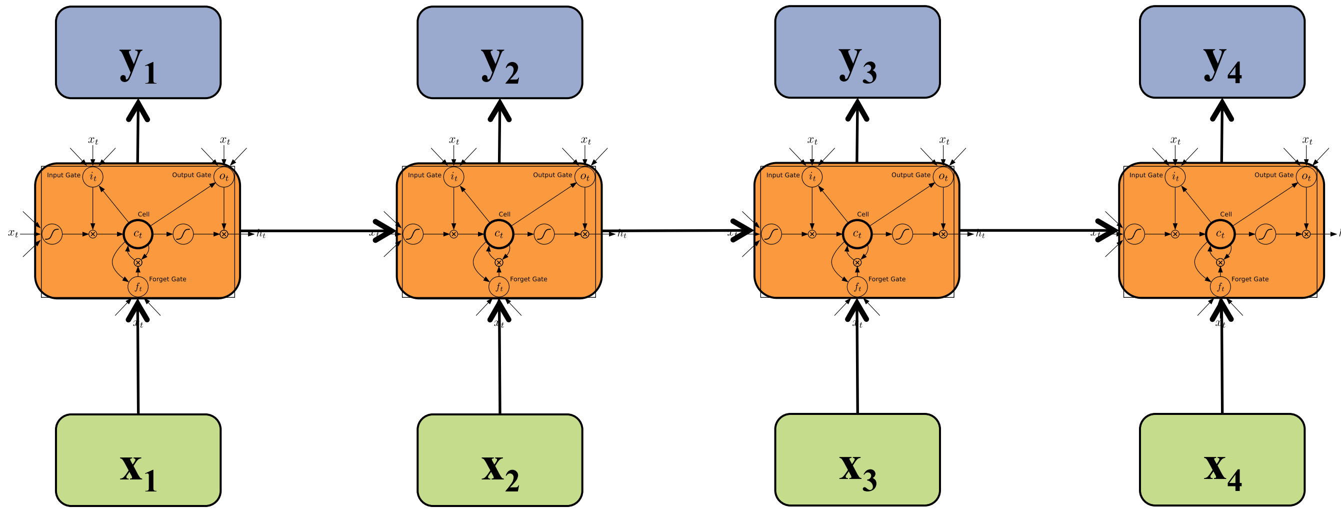
$$c_t = f_t c_{t-1} + i_t \tanh\left(W_{xc}x_t + W_{hc}h_{t-1} + b_c\right)$$

$$o_t = \sigma\left(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o\right)$$

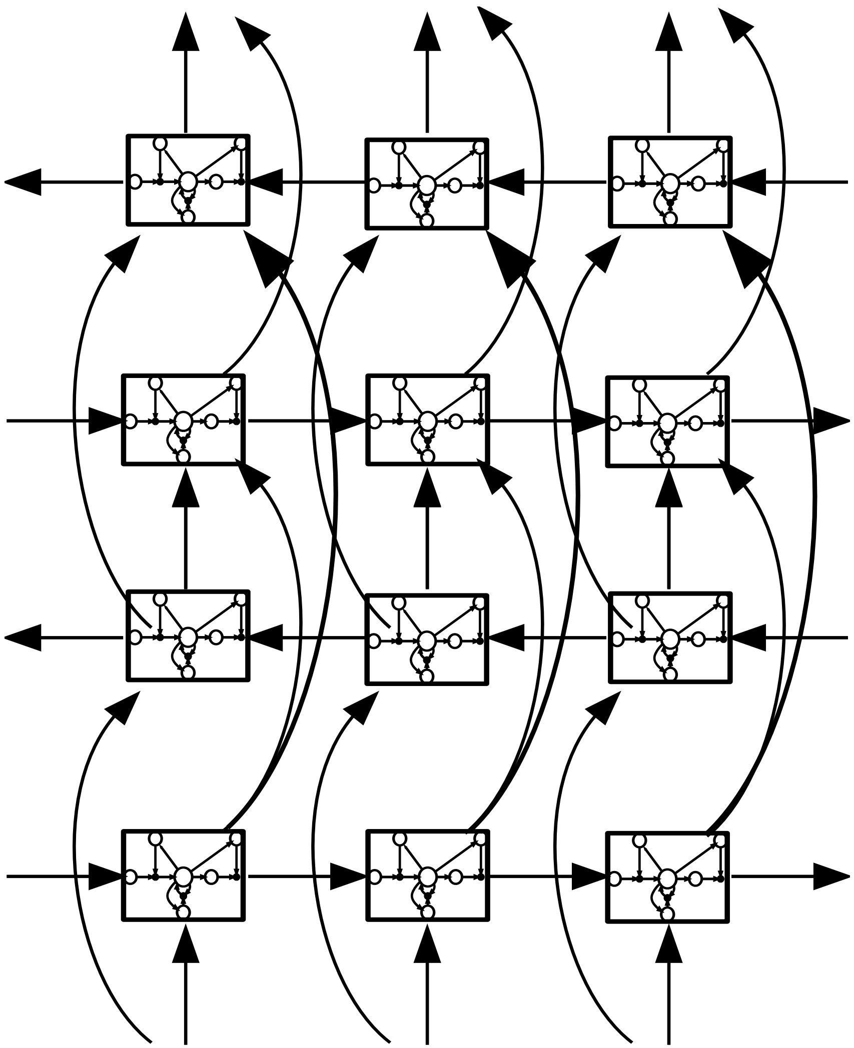
$$h_t = o_t \tanh(c_t)$$

Figure from (Graves et al., 2013)

Long Short-Term Memory (LSTM)



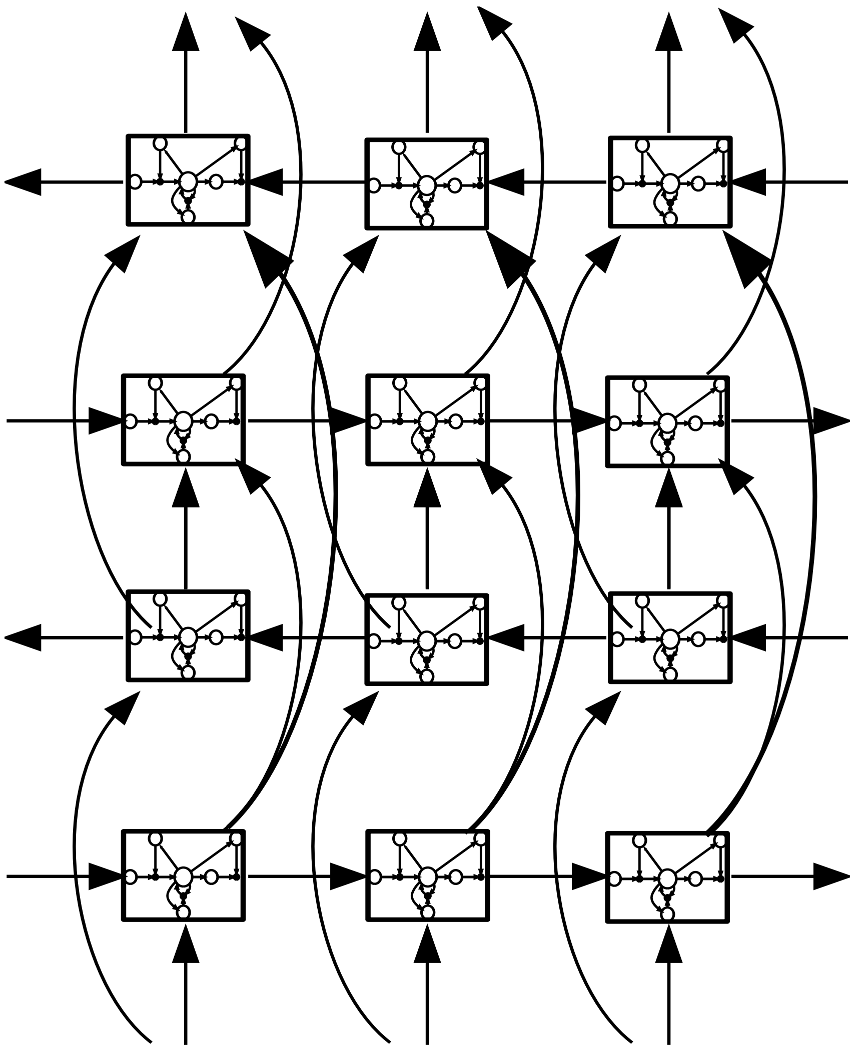
Deep Bidirectional LSTM (DBLSTM)



- Figure: input/output layers not shown
- **Same general topology** as a Deep Bidirectional RNN, but with **LSTM units** in the hidden layers
- No additional **representational power** over DBRNN, but **easier to learn** in practice

Figure from (Graves et al., 2013)

Deep Bidirectional LSTM (DBLSTM)



How important is this particular architecture?

Jozefowicz et al. (2015) **evaluated 10,000 different LSTM-like architectures** and found several variants that worked just as well on several tasks.

Figure from (Graves et al., 2013)

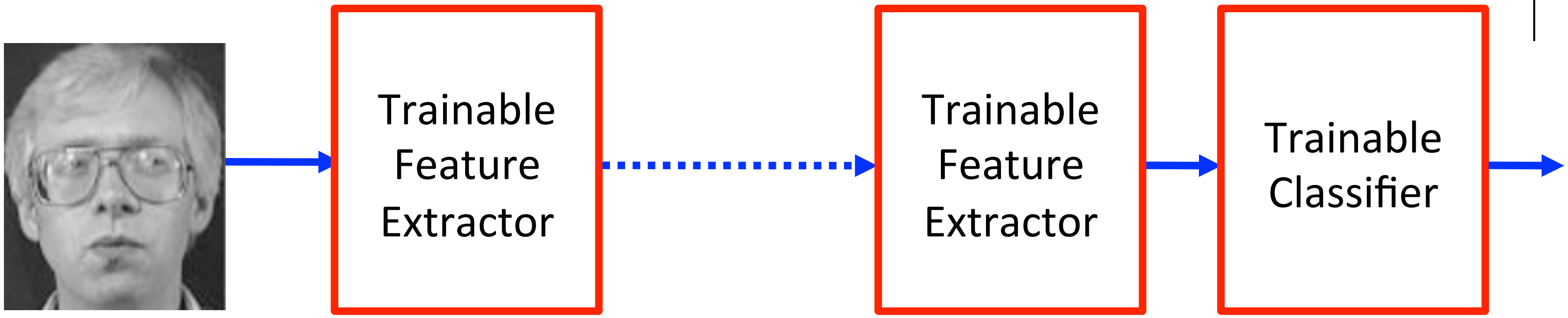
CONVOLUTIONAL NEURAL NETS

Expressive Capabilities of ANNs

- Boolean functions:
  - Every Boolean function can be represented by network with single hidden layer
  - But might require exponential (in number of inputs) hidden units

- Continuous functions:
  - Every bounded continuous function can be approximated with arbitrary small error, by network with one hidden layer [Cybenko 1989; Hornik et al 1989]
  - Any function can be approximated to arbitrary accuracy by a network with two hidden layers [Cybenko 1988].

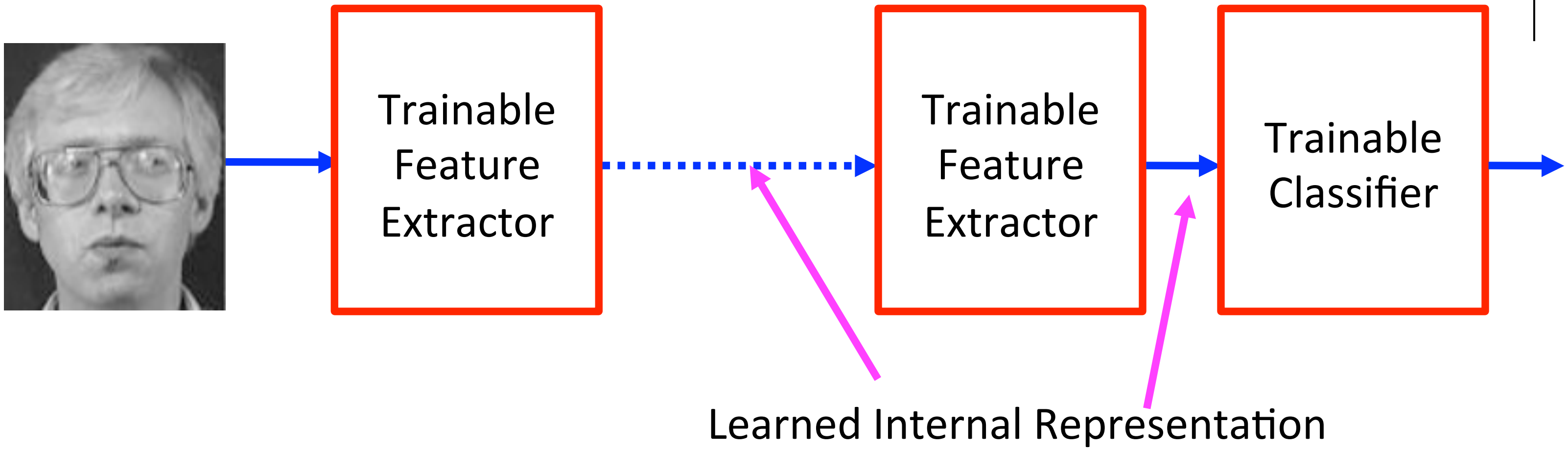
Using ANN to hierarchical representation



Good Representations are hierarchical

- In Language: hierarchy in syntax and semantics
  - Words->Parts of Speech->Sentences->Text
  - Objects,Actions,Attributes...-> Phrases -> Statements -> Stories
- In Vision: part-whole hierarchy
  - Pixels->Edges->Textons->Parts->Objects->Scenes

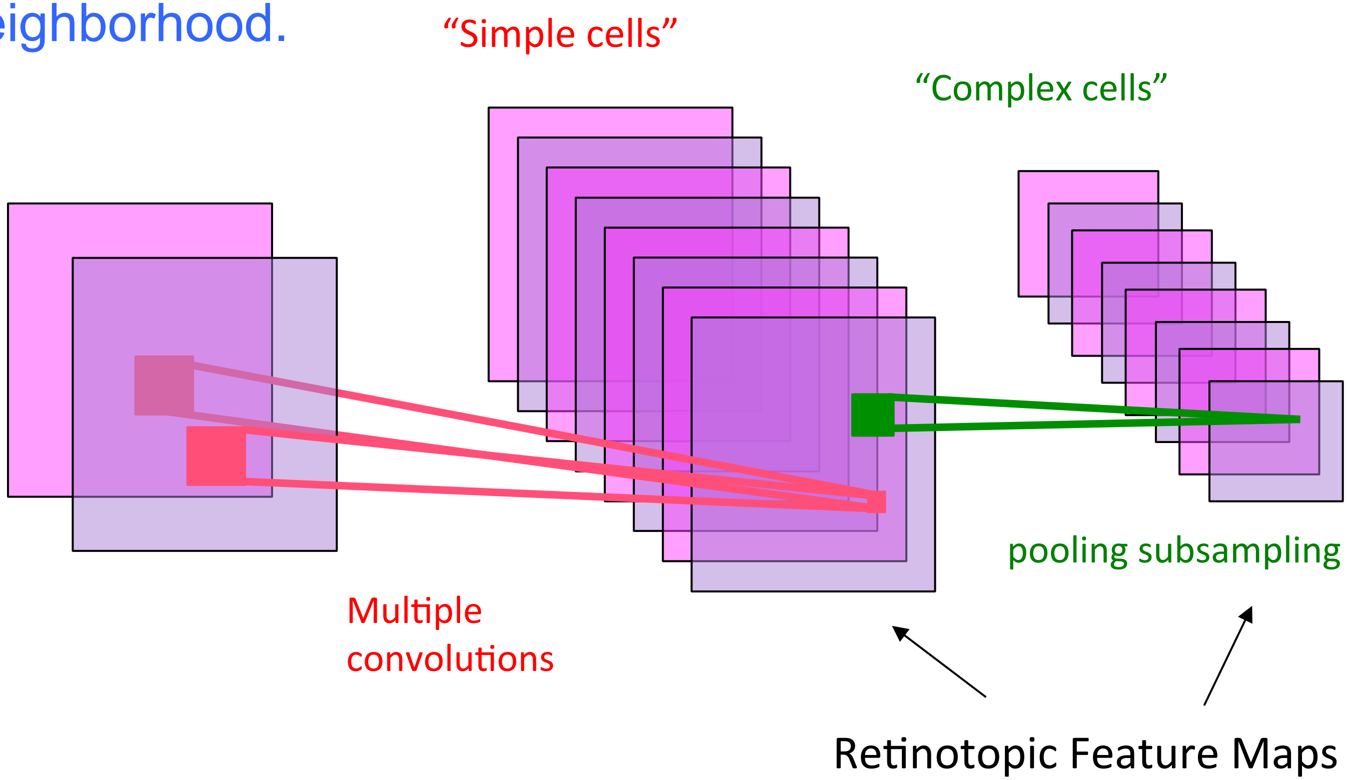
"Deep" learning: learning hierarchical representations



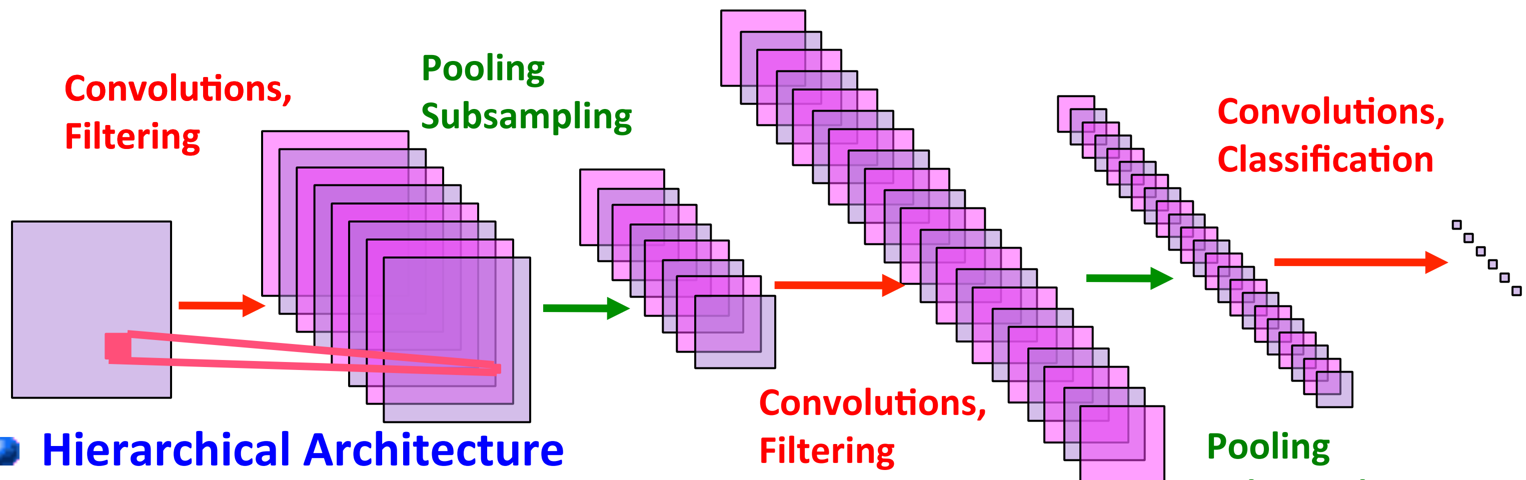
- Deep Learning: learning a hierarchy of internal representations
- From low-level features to mid-level invariant representations, to object identities
- Representations are increasingly invariant as we go up the layers
- using multiple stages gets around the specificity/invariance dilemma

Filtering+NonLinearity+Pooling = 1 stage of a Convolutional Net

- [Hubel & Wiesel 1962]:
  - simple cells detect local features
  - complex cells "pool" the outputs of simple cells within a retinotopic neighborhood.

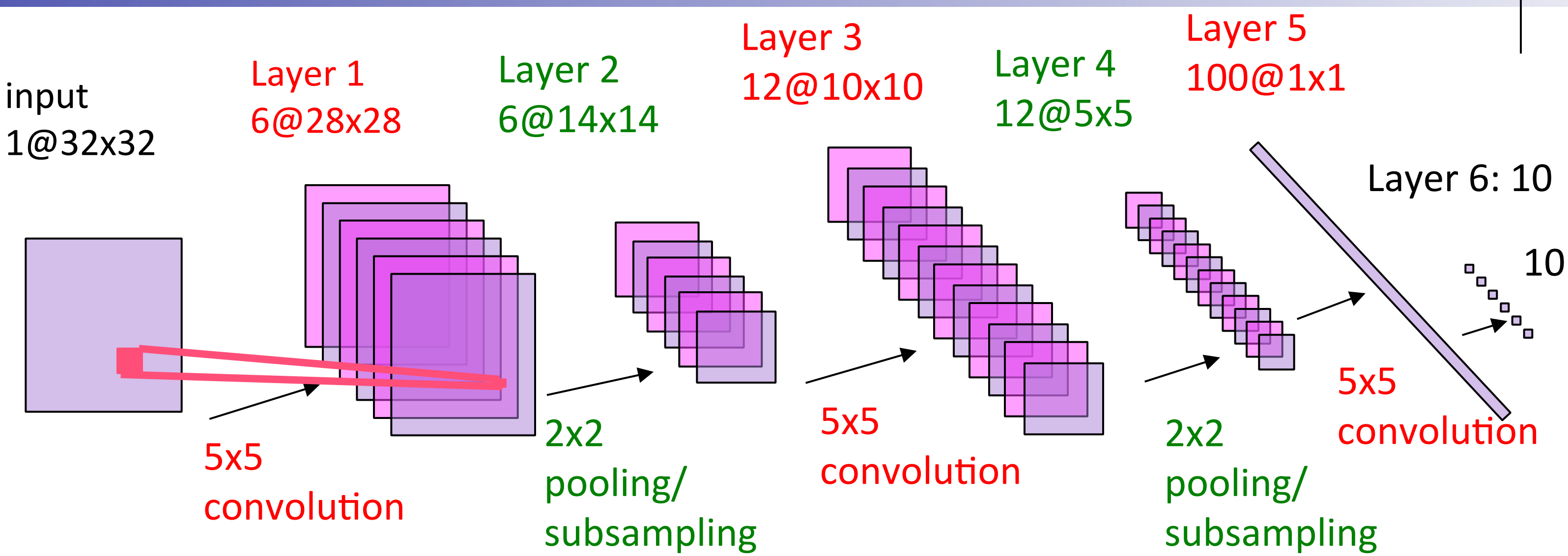


Convolutional Network: Multi-Stage Trainable Architecture



- **Hierarchical Architecture**
  - Representations are more global, more invariant, and more abstract as we go up the layers
- **Alternated Layers of Filtering and Spatial Pooling**
  - Filtering detects conjunctions of features
  - Pooling computes local disjunctions of features
- **Fully Trainable**
  - All the layers are trainable

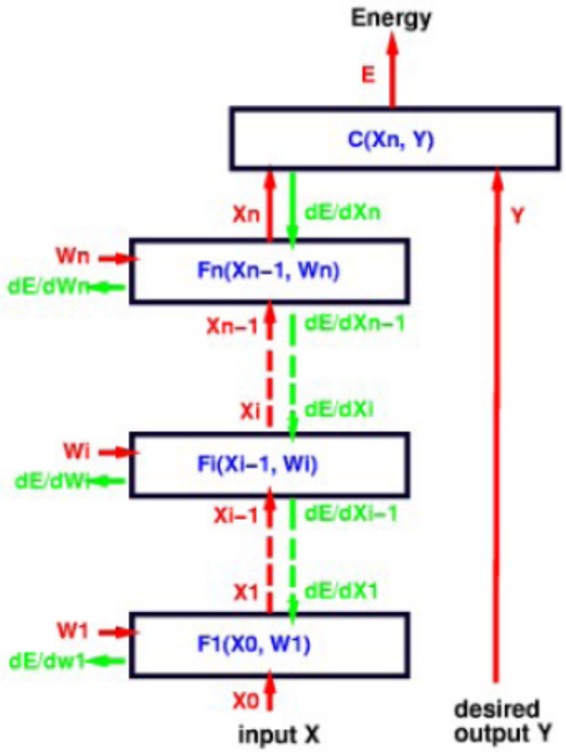
Convolutional Net Architecture for Hand-writing recognition



- Convolutional net for handwriting recognition (400,000 synapses)
  - Convolutional layers (simple cells): all units in a feature plane share the same weights
  - Pooling/subsampling layers (complex cells): for invariance to small distortions.
  - Supervised gradient-descent learning using back-propagation
  - The entire network is trained end-to-end. All the layers are trained simultaneously.
  - [LeCun et al. Proc IEEE, 1998]

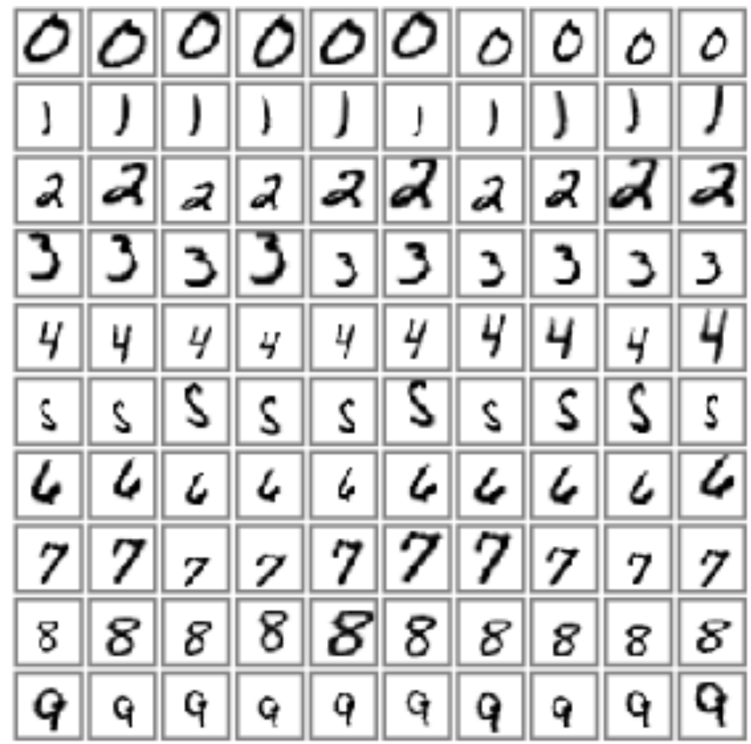
How to train?

To compute all the derivatives, we use a backward sweep called the **back-propagation algorithm** that uses the recurrence equation for $\frac{\partial E}{\partial X_i}$



- $\frac{\partial E}{\partial X_n} = \frac{\partial C(X_n, Y)}{\partial X_n}$
- $\frac{\partial E}{\partial X_{n-1}} = \frac{\partial E}{\partial X_n} \frac{\partial F_n(X_{n-1}, W_n)}{\partial X_{n-1}}$
- $\frac{\partial E}{\partial W_n} = \frac{\partial E}{\partial X_n} \frac{\partial F_n(X_{n-1}, W_n)}{\partial W_n}$
- $\frac{\partial E}{\partial X_{n-2}} = \frac{\partial E}{\partial X_{n-1}} \frac{\partial F_{n-1}(X_{n-2}, W_{n-1})}{\partial X_{n-2}}$
- $\frac{\partial E}{\partial W_{n-1}} = \frac{\partial E}{\partial X_{n-1}} \frac{\partial F_{n-1}(X_{n-2}, W_{n-1})}{\partial W_{n-1}}$
- ....etc, until we reach the first module.
- we now have all the $\frac{\partial E}{\partial W_i}$ for $i \in [1, n]$.

Application: MNIST Handwritten Digit Dataset



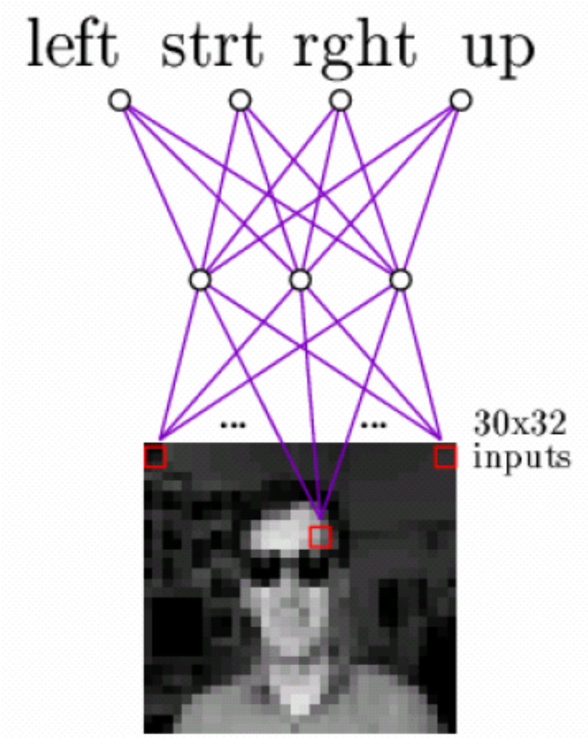
Handwritten Digit Dataset MNIST: 60,000 training samples, 10,000 test samples

Results on MNIST Handwritten Digits

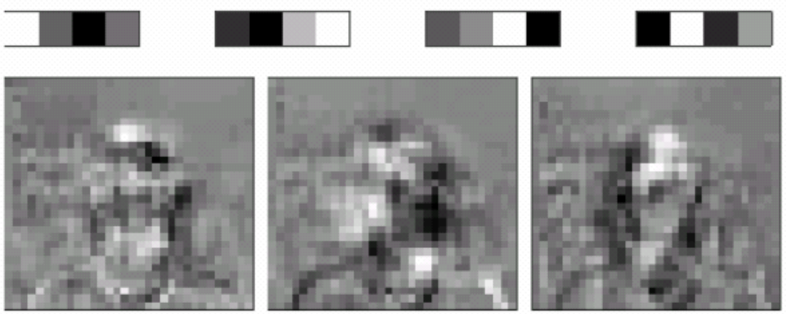
CLASSIFIER	DEFORMATION	PREPROCESSING	ERROR (%)	Reference
linear classifier (1-layer NN)		none	12.00	LeCun et al. 1998
linear classifier (1-layer NN)		deskewing	8.40	LeCun et al. 1998
pairwise linear classifier		deskewing	7.60	LeCun et al. 1998
K-nearest-neighbors, (L2)		none	3.09	Kenneth Wilder, U. Chicago
K-nearest-neighbors, (L2)		deskewing	2.40	LeCun et al. 1998
K-nearest-neighbors, (L2)		deskew, clean, blur	1.80	Kenneth Wilder, U. Chicago
K-NN L3, 2 pixel jitter		deskew, clean, blur	1.22	Kenneth Wilder, U. Chicago
K-NN, shape context matching		shape context feature	0.63	Belongie et al. IEEE PAMI 2002
40 PCA + quadratic classifier		none	3.30	LeCun et al. 1998
1000 RBF + linear classifier		none	3.60	LeCun et al. 1998
K-NN, Tangent Distance		subsamp 16x16 pixels	1.10	LeCun et al. 1998
SVM, Gaussian Kernel		none	1.40	
SVM deg 4 polynomial		deskewing	1.10	LeCun et al. 1998
Reduced Set SVM deg 5 poly		deskewing	1.00	LeCun et al. 1998
Virtual SVM deg-9 poly	Affine	none	0.80	LeCun et al. 1998
V-SVM, 2-pixel jittered		none	0.68	DeCoste and Scholkopf, MLJ 2002
V-SVM, 2-pixel jittered		deskewing	0.56	DeCoste and Scholkopf, MLJ 2002
2-layer NN, 300 HU, MSE		none	4.70	LeCun et al. 1998
2-layer NN, 300 HU, MSE,	Affine	none	3.60	LeCun et al. 1998
2-layer NN, 300 HU		deskewing	1.60	LeCun et al. 1998
3-layer NN, 500+150 HU		none	2.95	LeCun et al. 1998
3-layer NN, 500+150 HU	Affine	none	2.45	LeCun et al. 1998
3-layer NN, 500+300 HU, CE, reg		none	1.53	Hinton, unpublished, 2005
2-layer NN, 800 HU, CE		none	1.60	Simard et al., ICDAR 2003
2-layer NN, 800 HU, CE	Affine	none	1.10	Simard et al., ICDAR 2003
2-layer NN, 800 HU, MSE	Elastic	none	0.90	Simard et al., ICDAR 2003
2-layer NN, 800 HU, CE	Elastic	none	0.70	Simard et al., ICDAR 2003
Convolutional net LeNet-1		subsamp 16x16 pixels	1.70	LeCun et al. 1998
Convolutional net LeNet-4		none	1.10	LeCun et al. 1998
Convolutional net LeNet-5,		none	0.95	LeCun et al. 1998
Conv. net LeNet-5,	Affine	none	0.80	LeCun et al. 1998
Boosted LeNet-4	Affine	none	0.70	LeCun et al. 1998
Conv. net, CE	Affine	none	0.60	Simard et al., ICDAR 2003
Comv net, CE	Elastic	none	0.40	Simard et al., ICDAR 2003

Application: ANN for Face Reco.

- The model



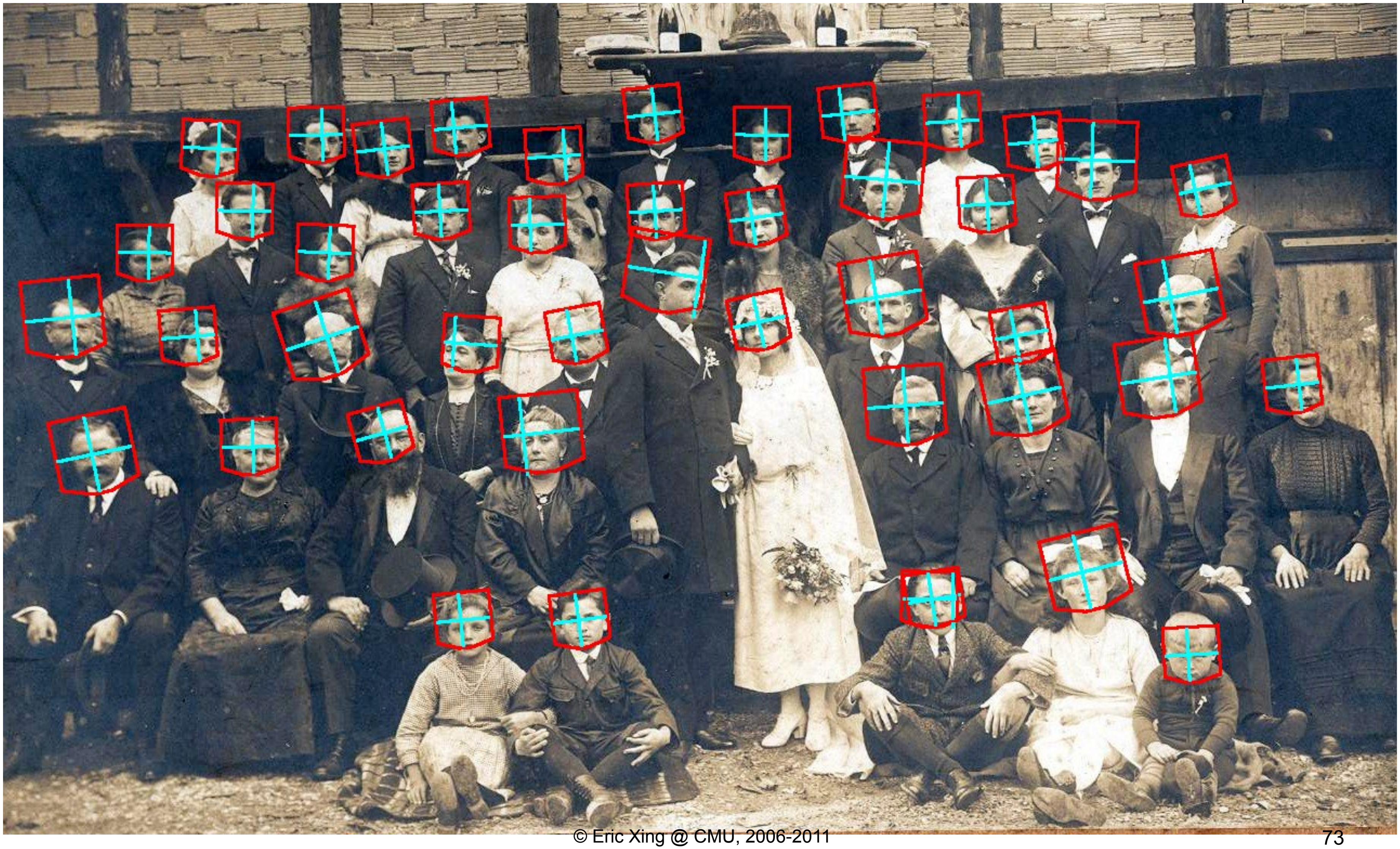
- The learned hidden unit weights



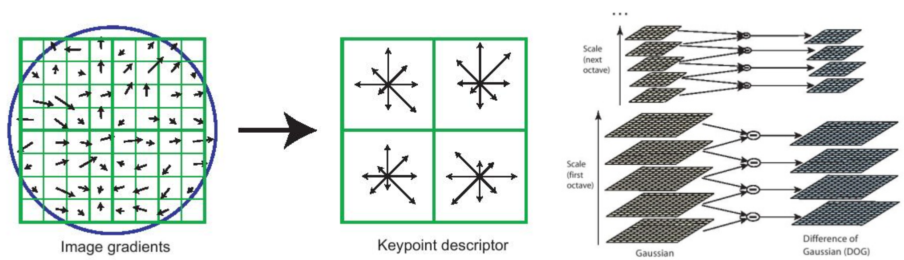
Typical input images

http://www.cs.cmu.edu/~tom/faces.html

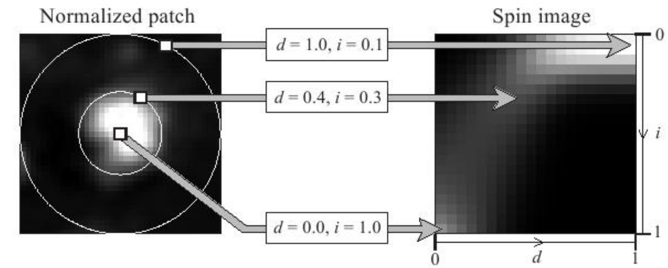
Face Detection with a Convolutional Net



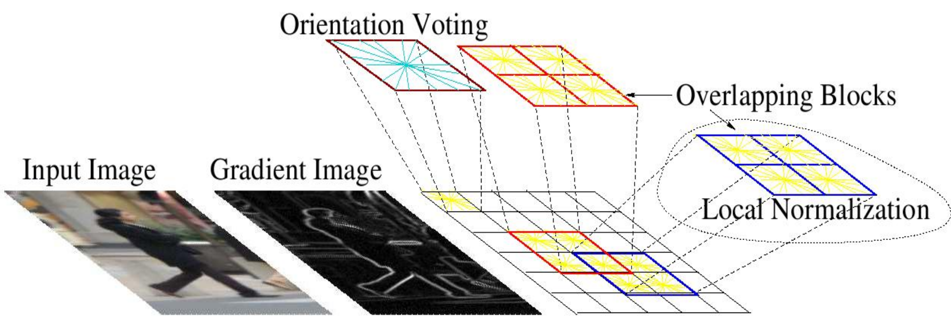
Computer vision features

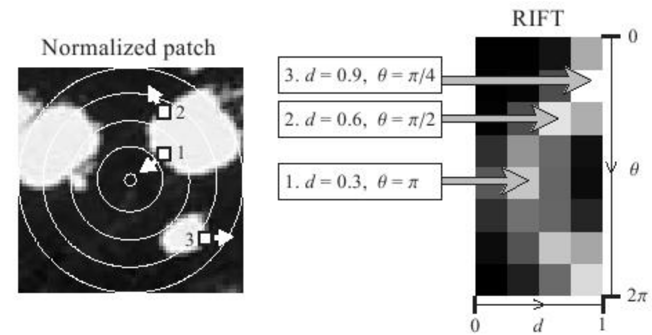


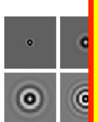
SIFT



Spin image

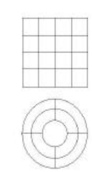






Drawbacks of feature engineering

1. Needs expert knowledge
2. Time consuming hand-tuning



(e)

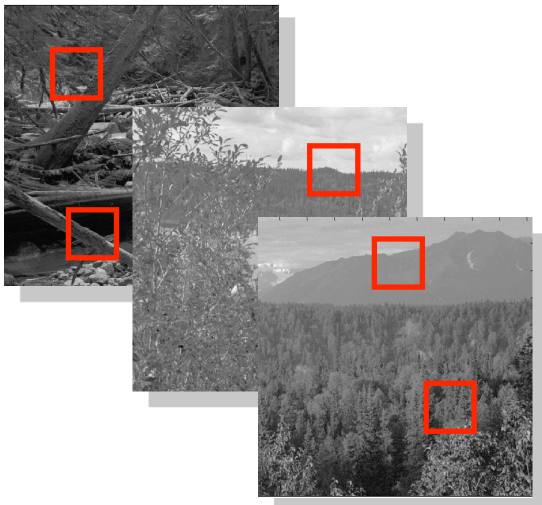
Textons

GLOH

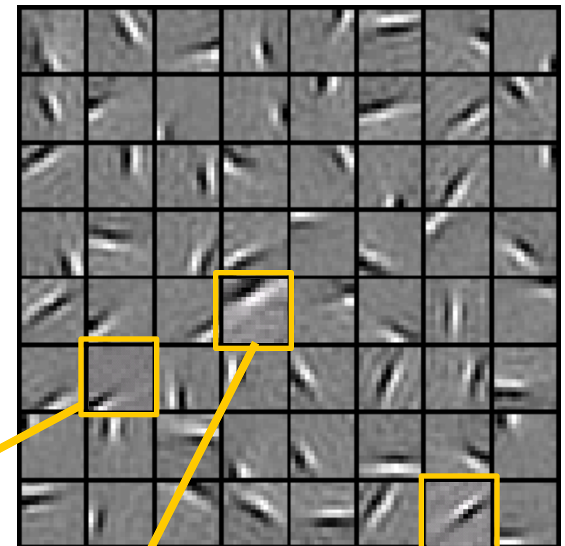
and Ng

Sparse coding on images

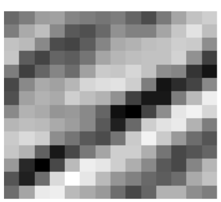
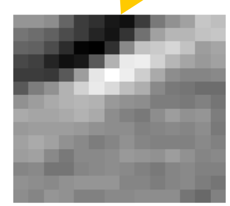
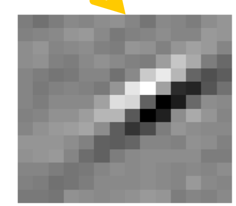
Natural Images



Learned bases: "Edges"



New example

 $= 0.8 *$  $+ 0.3 *$  $+ 0.5 *$ 

$$x = 0.8 * b_{36} + 0.3 * b_{42} + 0.5 * b_{65}$$

[0, 0, … **0.8**, …, **0.3**, …, **0.5**, …] = coefficients (feature representation)

Courtesy: Lee and Ng

Basis (or features) can be learned by Optimization

Given input data $\{x^{(1)}, \ldots, x^{(m)}\}$, we want to find good bases $\{b_1, \ldots, b_n\}$:

$$\min_{b,a} \sum_i \| x^{(i)} - \sum_j a_j^{(i)} b_j \|_2^2 + \beta \sum_i \| a^{(i)} \|_1$$

Reconstruction error — Sparsity penalty

$$\forall j: \ \| b_j \| \le 1$$

Normalization constraint

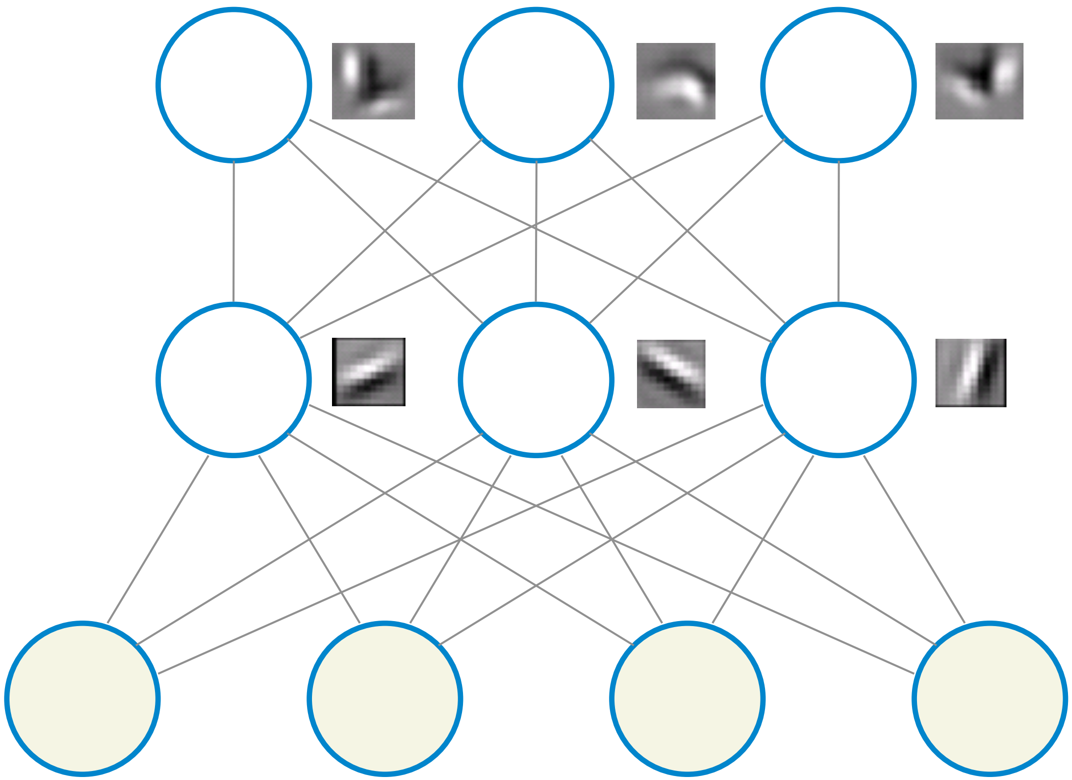
Solve by alternating minimization:

-- Keep *b* fixed, find optimal *a*.

-- Keep *a* fixed, find optimal *b*.

Courtesy: Lee and Ng

Learning Feature Hierarchy

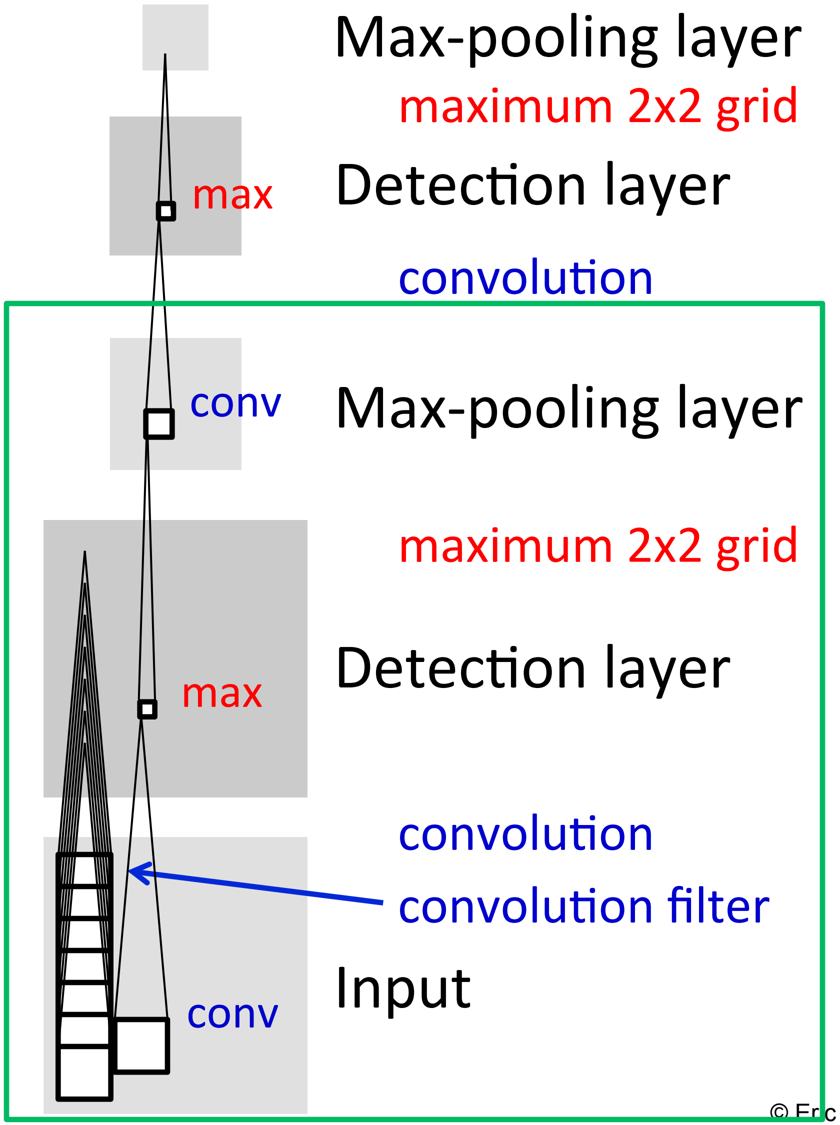


DBN (Hinton et al., 2006) with additional sparseness constraint.

[Related work: Hinton, Bengio, LeCun, and others.]

Courtesy: Lee and Ng

Convolutional architectures



- Weight sharing by convolution (e.g., [Lecun et al., 1989])
- "Max-pooling"
  Invariance
  Computational efficiency
  Deterministic and feed-forward
- One can develop convolutional Restricted Boltzmann machine (CRBM).
- One can define *probabilistic max-pooling* that *combine bottom-up and top-down information*.

Courtesy: Lee and Ng

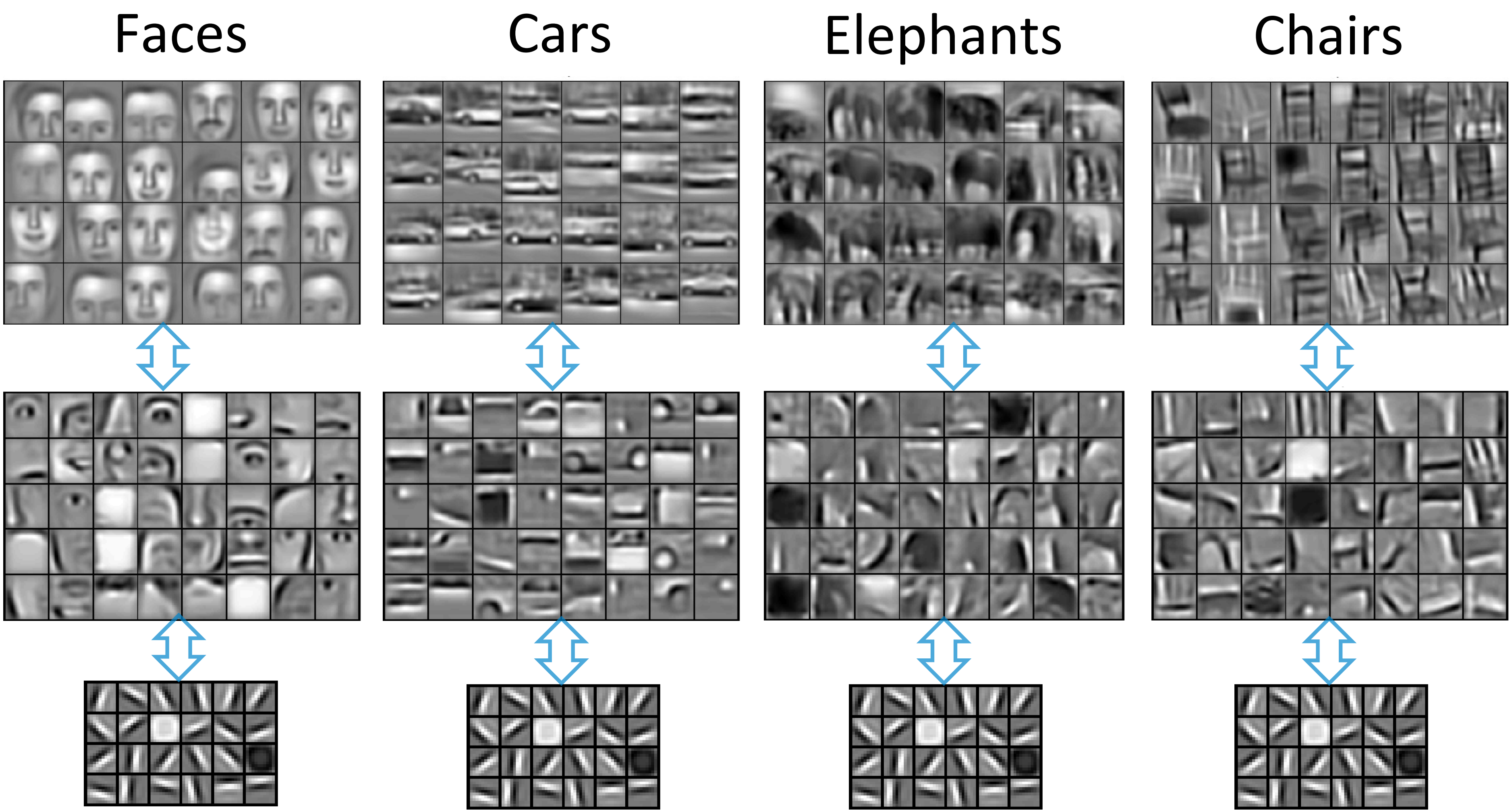
Convolutional Deep Belief Networks

- Bottom-up (greedy), layer-wise training
  - Train one layer (convolutional RBM) at a time.

- Inference (approximate)
  - Undirected connections for all layers (Markov net) [Related work: Salakhutdinov and Hinton, 2009]
  - Block Gibbs sampling or mean-field
  - Hierarchical probabilistic inference

Courtesy: Lee and Ng

Unsupervised learning of object-parts



Courtesy: Lee and Ng

Weaknesses & Criticisms

- Learning everything. Better to encode prior knowledge about structure of images.

  A: Compare with machine learning vs. linguists debate in NLP.

- Results not yet competitive with best engineered systems.

  A: Agreed. True for some domains.

Courtesy: Lee and Ng

Tutorials

- LSTMs
  - Christopher Olah's blog
  - http://colah.github.io/posts/2015-08-Understanding-LSTMs/
- Convolutional Neural Networks
  - Andrej Karpathy, CS231n Notes
  - http://cs231n.github.io/convolutional-networks/