10707
Deep Learning

Russ Salakhutdinov

Machine Learning Department
rsalakhu@cs.cmu.edu

Deep Belief Networks

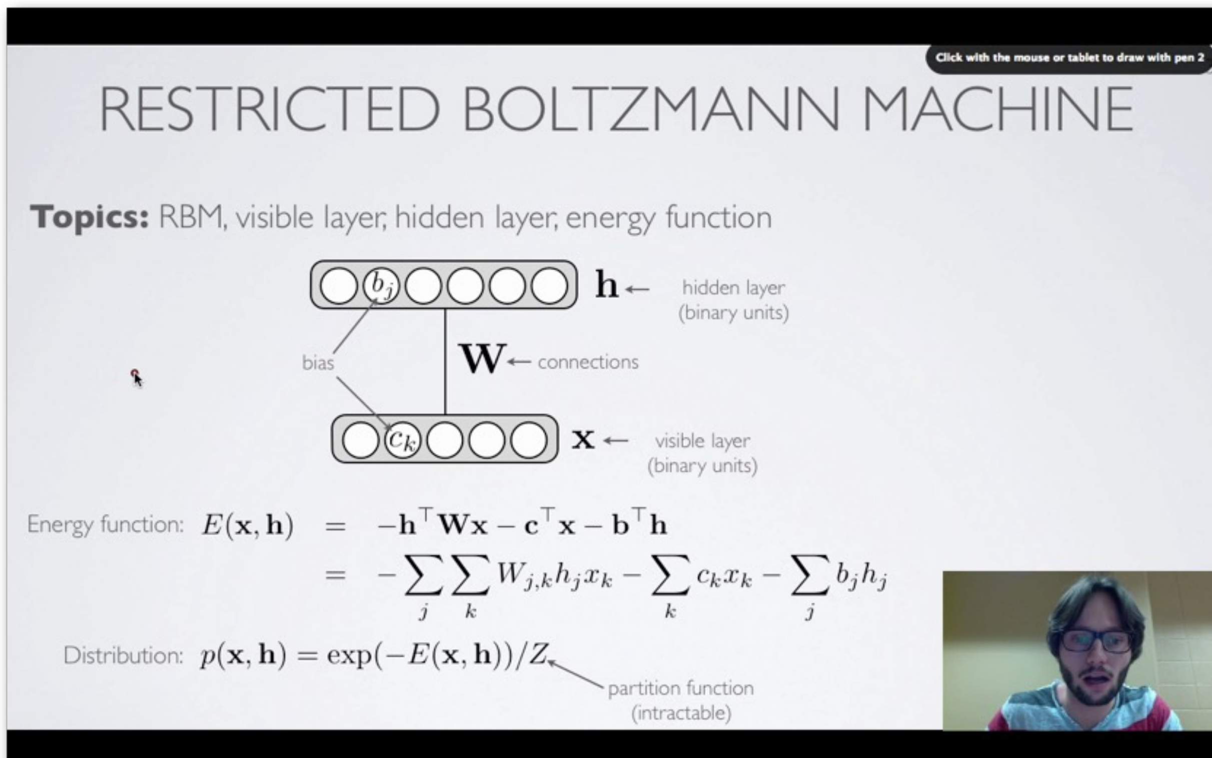
Neural Networks Online Course

- **Disclaimer**: Much of the material and slides for this lecture were borrowed from Hugo Larochelle's class on Neural Networks: https://sites.google.com/site/deeplearningsummerschool2016/

http://info.usherbrooke.ca/hlarochelle/neural_networks

- Hugo's class covers many other topics: convolutional networks, neural language model, Boltzmann machines, autoencoders, sparse coding, etc.

- We will use his material for some of the other lectures.



Multilayer Neural Net

- Consider a network with L hidden layers.
  - layer pre-activation for k>0

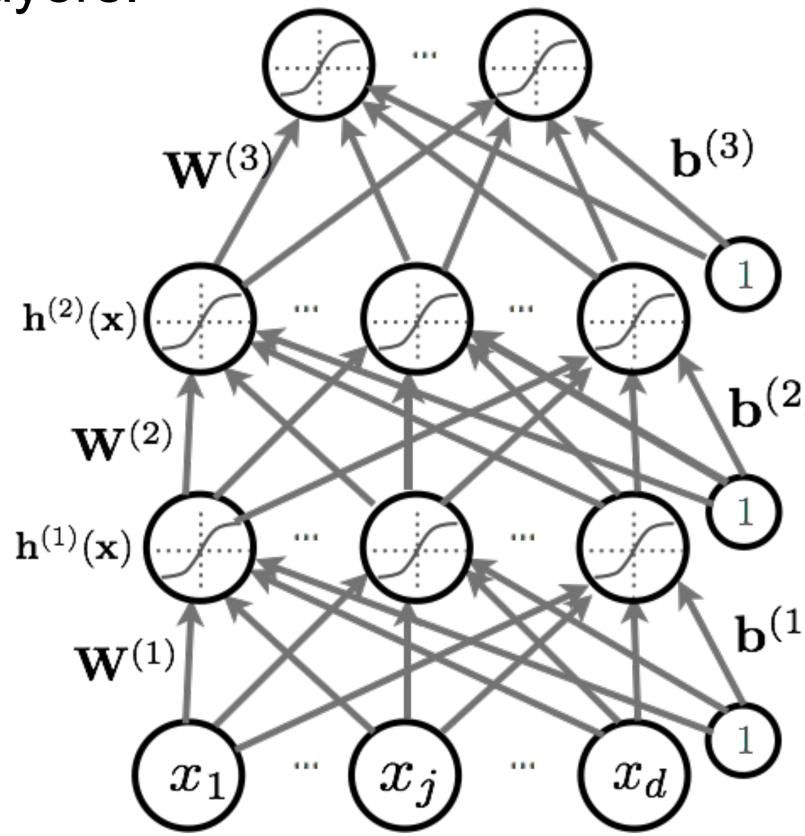
$$\mathbf{a}^{(k)}(\mathbf{x}) = \mathbf{b}^{(k)} + \mathbf{W}^{(k)}\mathbf{h}^{(k-1)}(\mathbf{x})$$

  - hidden layer activation from 1 to L:

$$\mathbf{h}^{(k)}(\mathbf{x}) = \mathbf{g}(\mathbf{a}^{(k)}(\mathbf{x}))$$

  - output layer activation (k=L+1):

$$\mathbf{h}^{(L+1)}(\mathbf{x}) = \mathbf{o}(\mathbf{a}^{(L+1)}(\mathbf{x})) = \mathbf{f}(\mathbf{x})$$

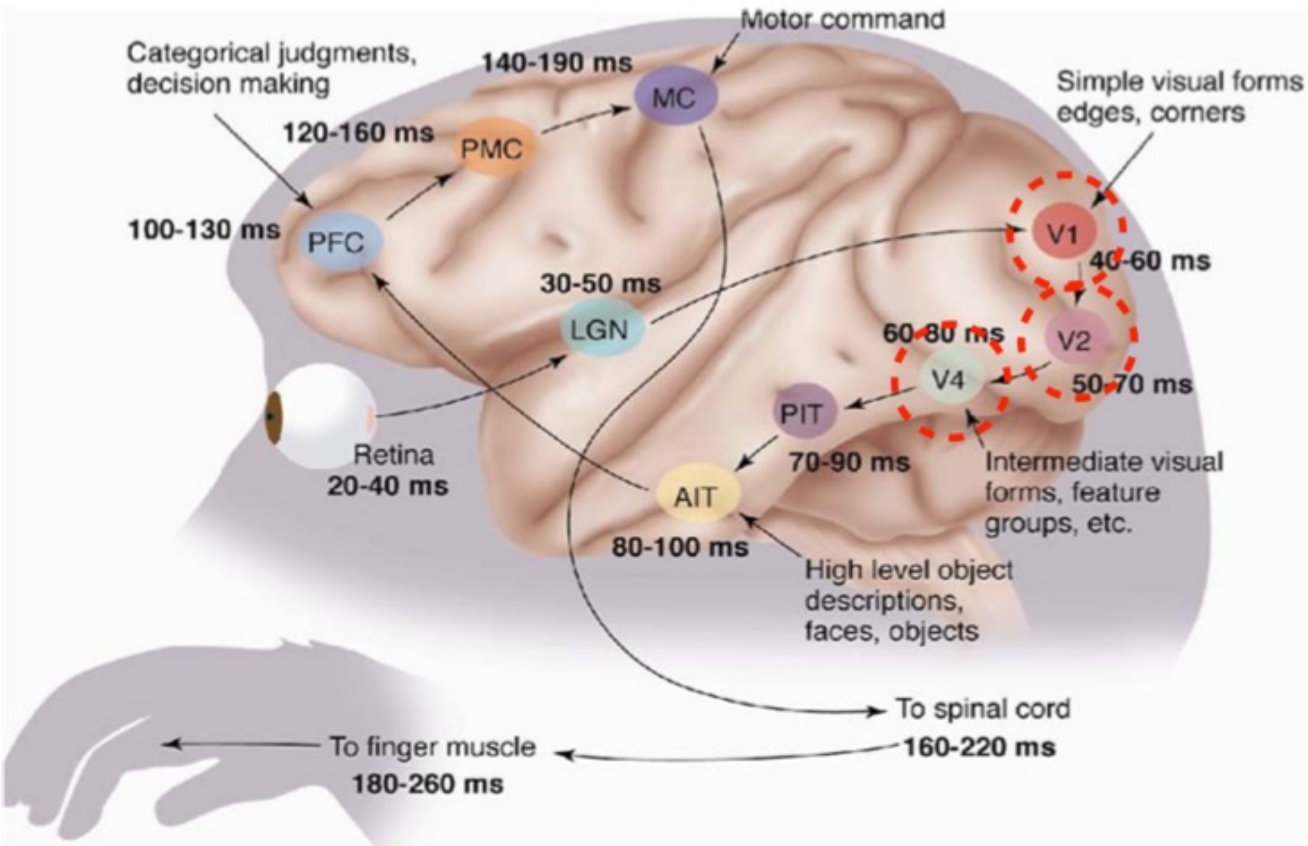


$$(\mathbf{h}^{(0)}(\mathbf{x}) = \mathbf{x})$$

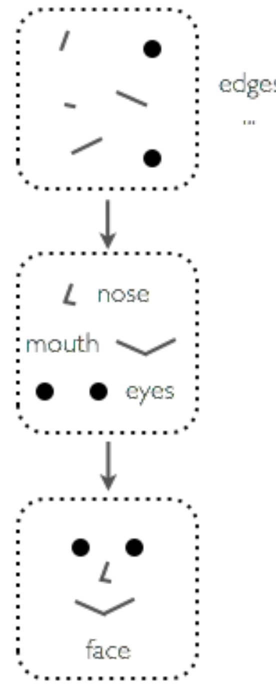
Learning Distributed Representations

- Deep learning is research on learning models with multilayer representations
  - multilayer (feed-forward) neural networks
  - multilayer graphical model (deep belief network, deep Boltzmann machine)
- Each layer learns "distributed representation"
  - Units in a layer are not mutually exclusive
    - each unit is a separate feature of the input
    - two units can be "active" at the same time
  - Units do not correspond to a partitioning (clustering) of the inputs
    - in clustering, an input can only belong to a single cluster

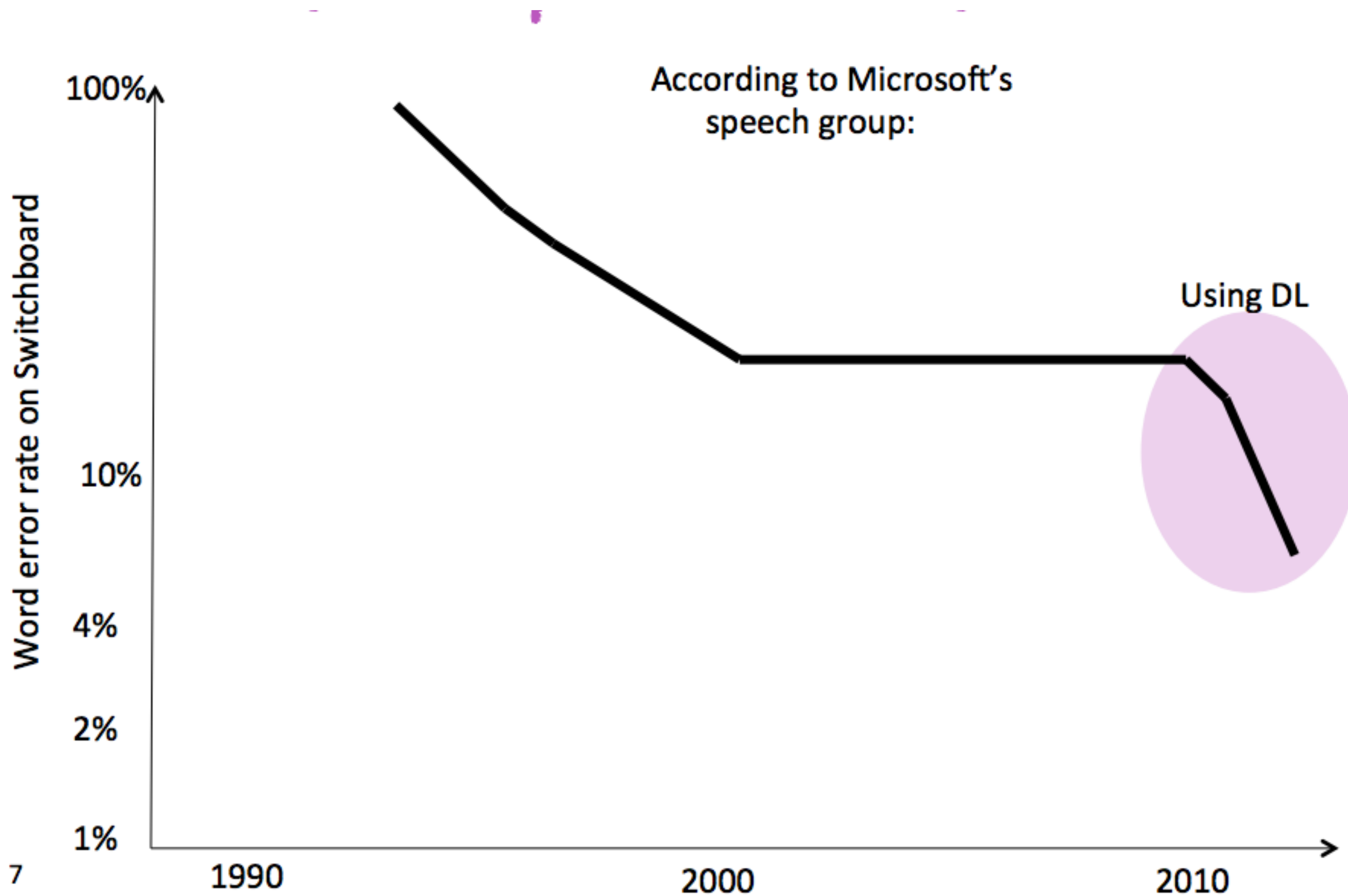
Inspiration from Visual Cortex



[picture from Simon Thorpe]

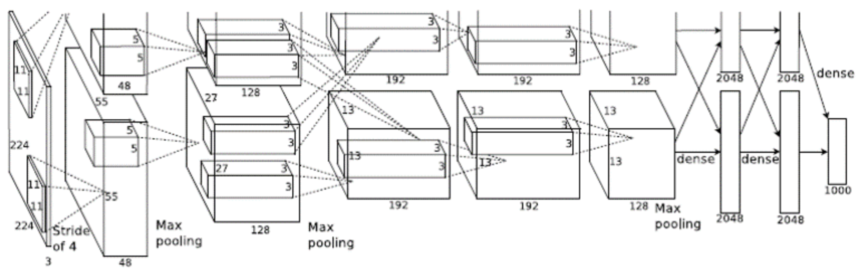


Success Story: Speech Recognition



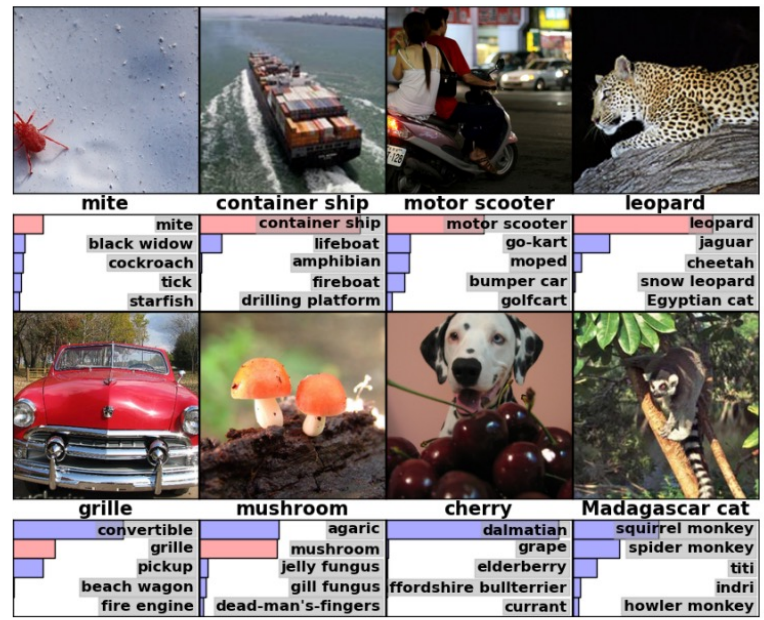
Success Story: Image Recognition

- Deep Convolutional Nets for Vision (Supervised)



IMAGENET

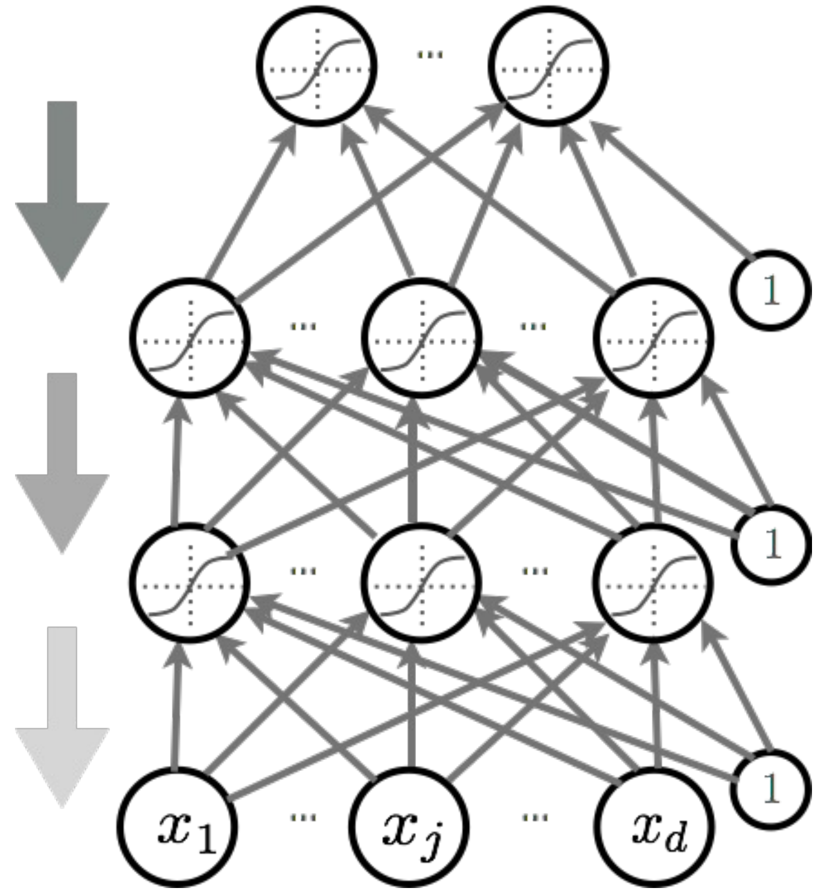
1.2 million training images
1000 classes



Why Training is Hard

- First hypothesis: Hard optimization problem (underfitting)
  - ➢ vanishing gradient problem
  - ➢ saturated units block gradient propagation

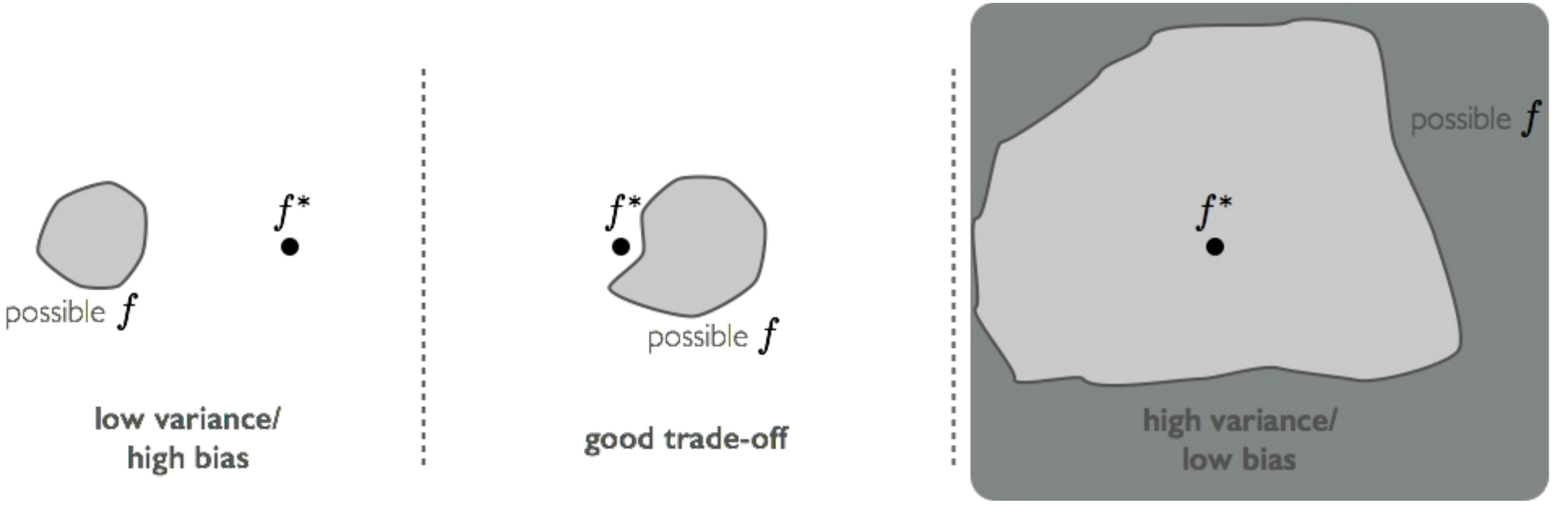
•This is a well known problem in recurrent neural networks



Why Training is Hard

- Second hypothesis: Overfitting
  - we are exploring a space of complex functions
  - deep nets usually have lots of parameters

- Might be in a high variance / low bias situation



Why Training is Hard

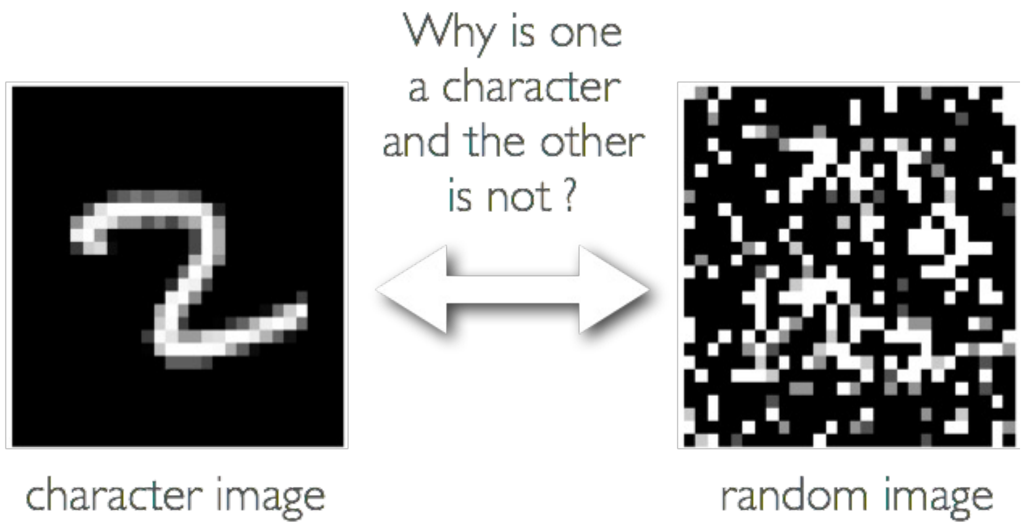
- First hypothesis (underfitting): better optimize
  - ➢ Use better optimization tools (e.g. batch-normalization, second order methods, such as KFAC)
  - ➢ Use GPUs, distributed computing.

- Second hypothesis (overfitting): use better regularization
  - ➢ Unsupervised pre-training
  - ➢ Stochastic drop-out training

- For many large-scale practical problems, you will need to use both: better optimization and better regularization!

Unsupervised Pre-training

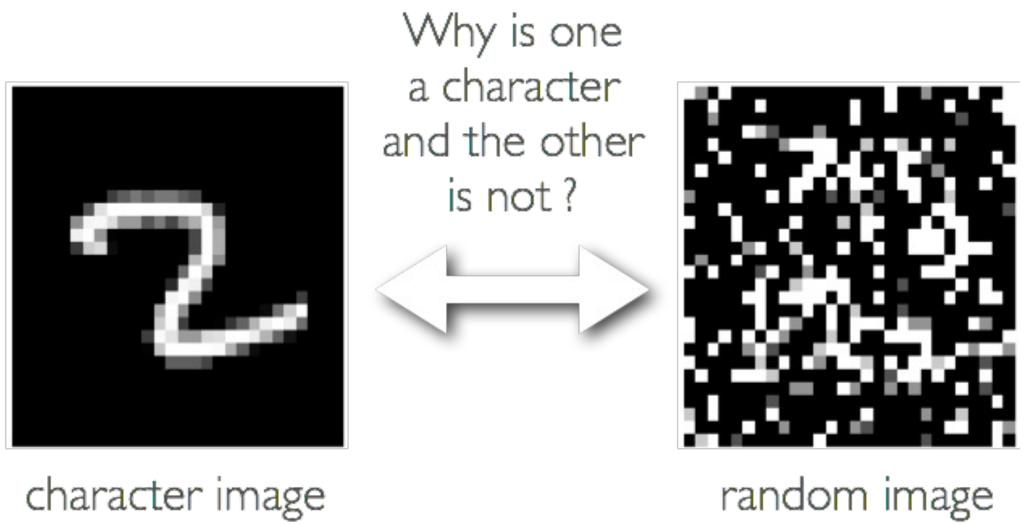
- Initialize hidden layers using unsupervised learning
  - ➢ Force network to represent latent structure of input distribution



  - ➢ Encourage hidden layers to encode that structure

Unsupervised Pre-training

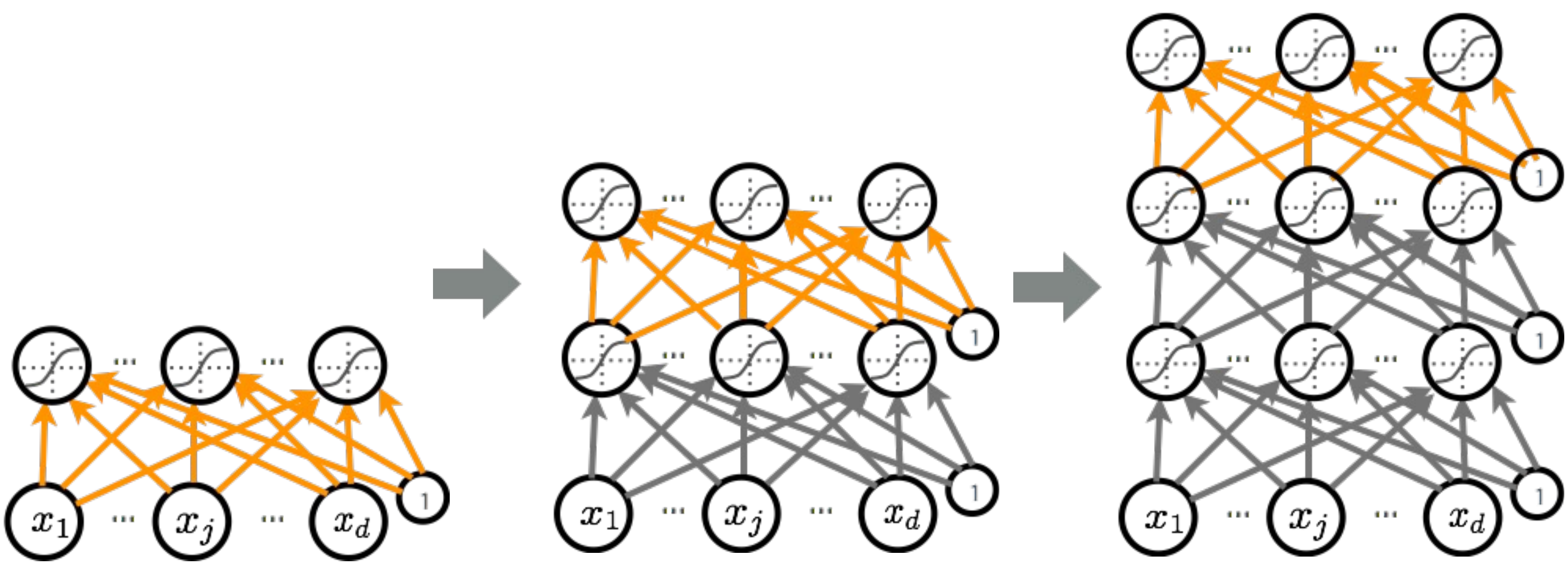
- Initialize hidden layers using unsupervised learning
  - This is a harder task than supervised learning (classification)



  - Hence we expect less overfitting

Pre-training

- We will use a greedy, layer-wise procedure
  - ➢ Train one layer at a time with unsupervised criterion
  - ➢ Fix the parameters of previous hidden layers
  - ➢ Previous layers viewed as feature extraction



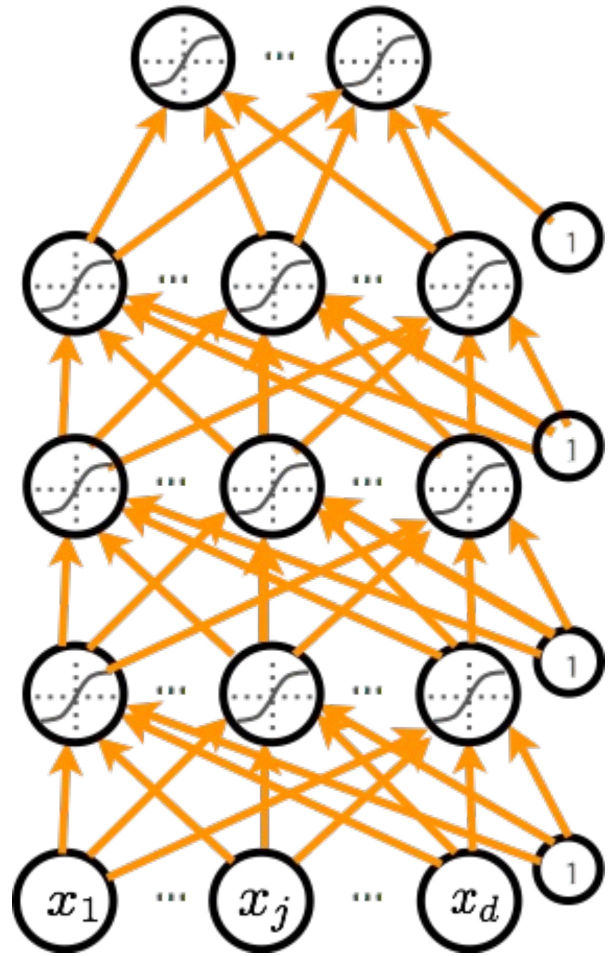
Pre-training

- Unsupervsed Pre-training
  - first layer: find hidden unit features that are more common in training inputs than in random inputs
  - second layer: find combinations of hidden unit features that are more common than random hidden unit features
  - third layer: find combinations of combinations of ...

- Pre-training initializes the parameters in a region such that the near local optima overfit less the data

Fine-tuning

- Once all layers are pre-trained
  - ➢ add output layer
  - ➢ train the whole network using supervised learning
- Supervised learning is performed as in a regular network
  - ➢ forward propagation, backpropagation and update
- We call this last phase fine-tuning
  - ➢ all parameters are "tuned" for the supervised task at hand
  - ➢ representation is adjusted to be more discriminative

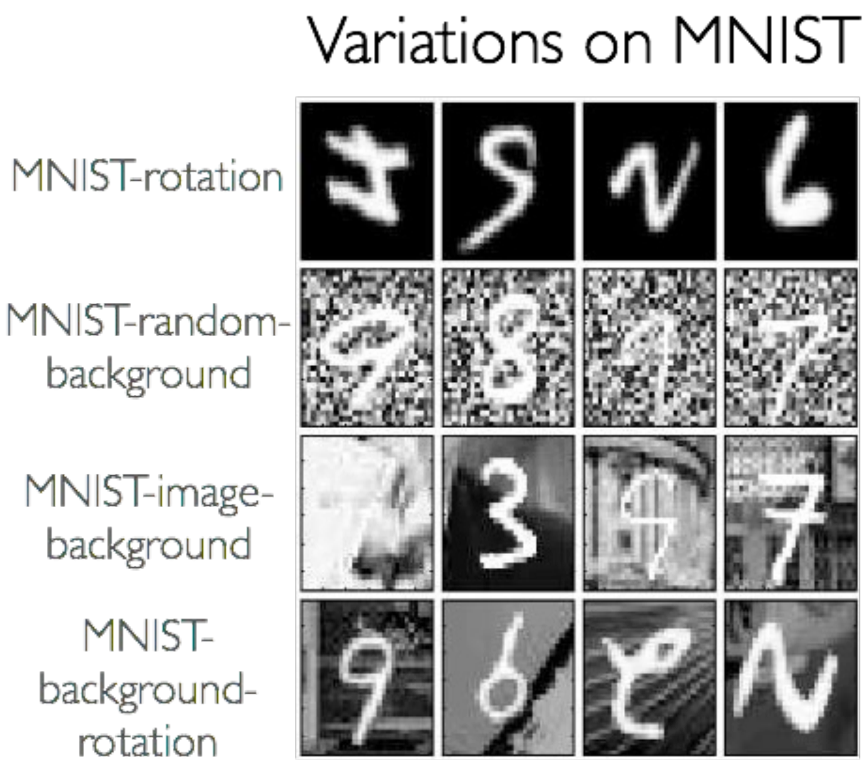


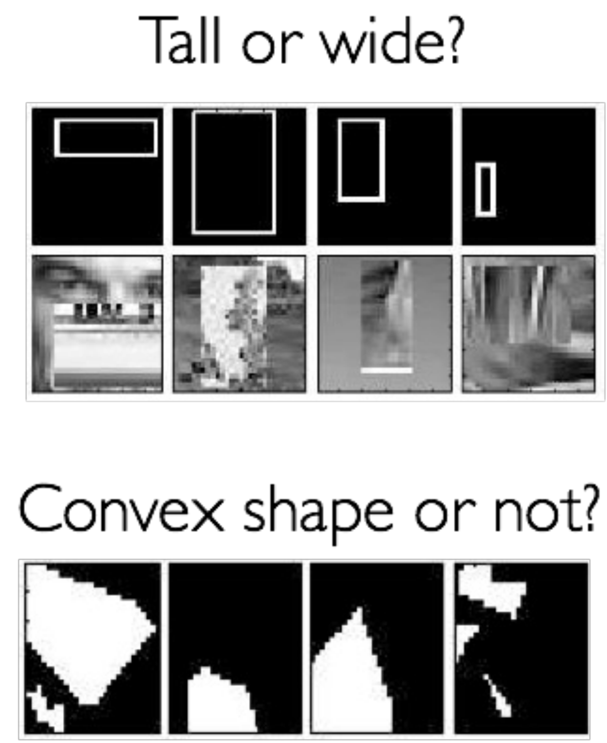
Stacking RBMs, Autoencoders

- Stacked Restricted Boltzmann Machines:
  - Hinton, Teh and Osindero suggested this procedure with RBMs,: **A fast learning algorithm for deep belief nets.**
  - To recognize shapes, first learn to generate images. Hinton, 2006.
- Stacked autoencoders, sparse-coding models, etc.
  - Bengio, Lamblin, Popovici and Larochelle (stacked autoencoders)
  - Ranzato, Poultney, Chopra and LeCun (stacked sparse coding models)
- Lots of others started stacking models together.

Example

- Datasets generated with varying number of factors of variations



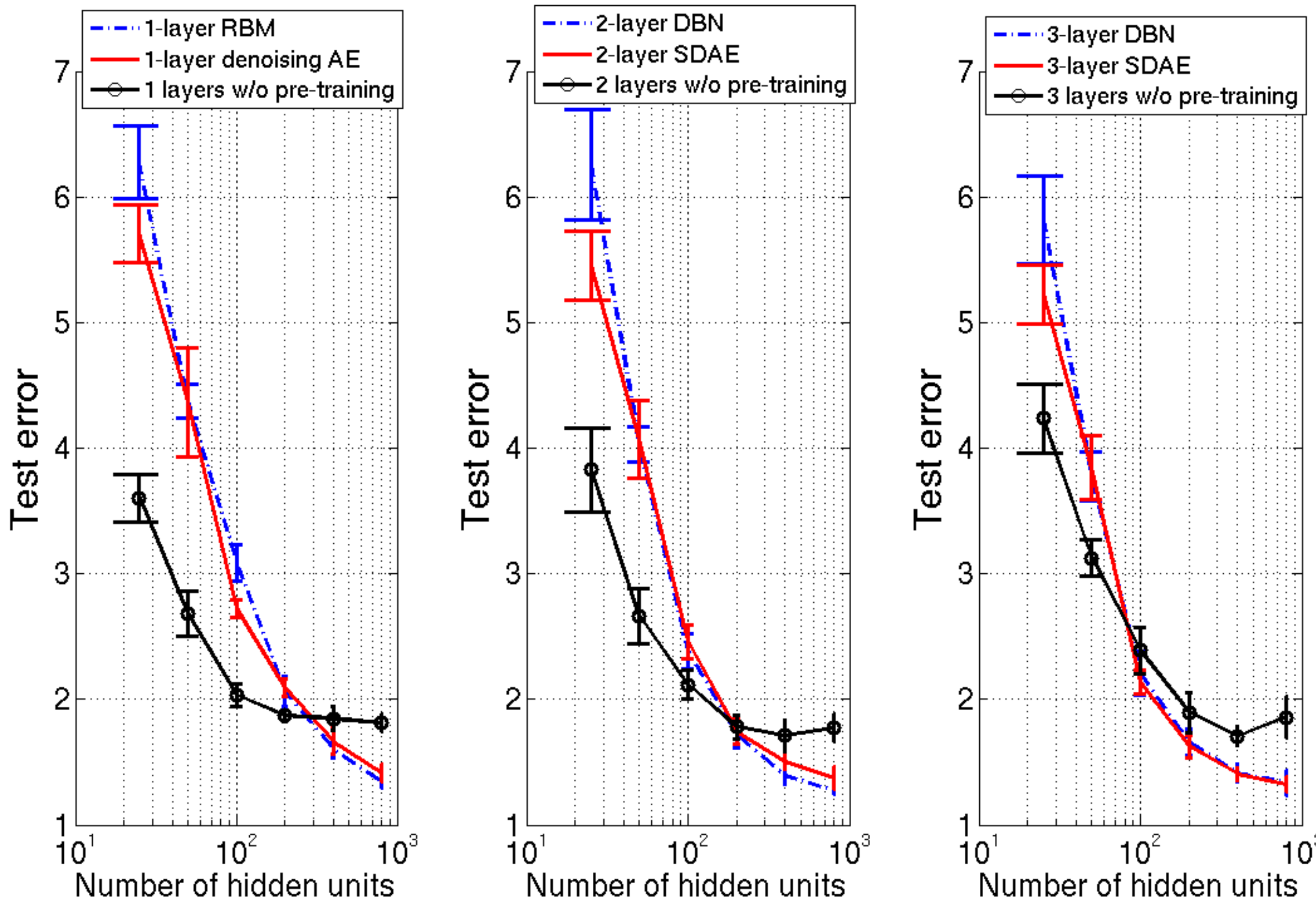


An Empirical Evaluation of Deep Architectures on Problems with Many Factors of Variation, Larochelle, Erhan, Courville, Bergstra and Bengio, 2007

Impact of Initialization

Network		MNIST-small classif. test error	MNIST-rotation classif. test error
Type	Depth		
Deep net	1	**4.14** % ± 0.17	15.22 % ± 0.31
	2	**4.03** % ± 0.17	**10.63** % ± 0.27
	3	**4.24** % ± 0.18	11.98 % ± 0.28
	4	4.47 % ± 0.18	11.73 % ± 0.29
Deep net + autoencoder	1	3.87 % ± 0.17	11.43% ± 0.28
	2	**3.38** % ± 0.16	9.88 % ± 0.26
	3	**3.37** % ± 0.16	**9.22** % ± 0.25
	4	**3.39** % ± 0.16	**9.20** % ± 0.25
Deep net + RBM	1	3.17 % ± 0.15	10.47 % ± 0.27
	2	**2.74** % ± 0.14	9.54 % ± 0.26
	3	**2.71** % ± 0.14	**8.80** % ± 0.25
	4	**2.72** % ± 0.14	**8.83** % ± 0.24

Impact of Pretraining



Acts as a regularizer: overfits less with large capacity, underfits with small capacity

Performance on Different Datasets

Stacked Autoencoders	Stacked RBMS	Stacked Denoising Autoencoders
SAA-3	**DBN-3**	**SdA-3** (ν)
3.46±0.16	3.11±0.15	**2.80±0.14** (10%)
10.30±0.27	**10.30±0.27**	**10.29±0.27** (10%)
11.28±0.28	**6.73±0.22**	10.38±0.27 (40%)
23.00±0.37	**16.31±0.32**	**16.68±0.33** (25%)
51.93±0.44	47.39±0.44	**44.49±0.44** (25%)
2.41±0.13	2.60±0.14	**1.99±0.12** (10%)
24.05±0.37	22.50±0.37	**21.59±0.36** (25%)
18.41±0.34	**18.63±0.34**	**19.06±0.34** (10%)

Extracting and Composing Robust Features with Denoising Autoencoders, Vincent, Larochelle, Bengio and Manzagol, 2008.

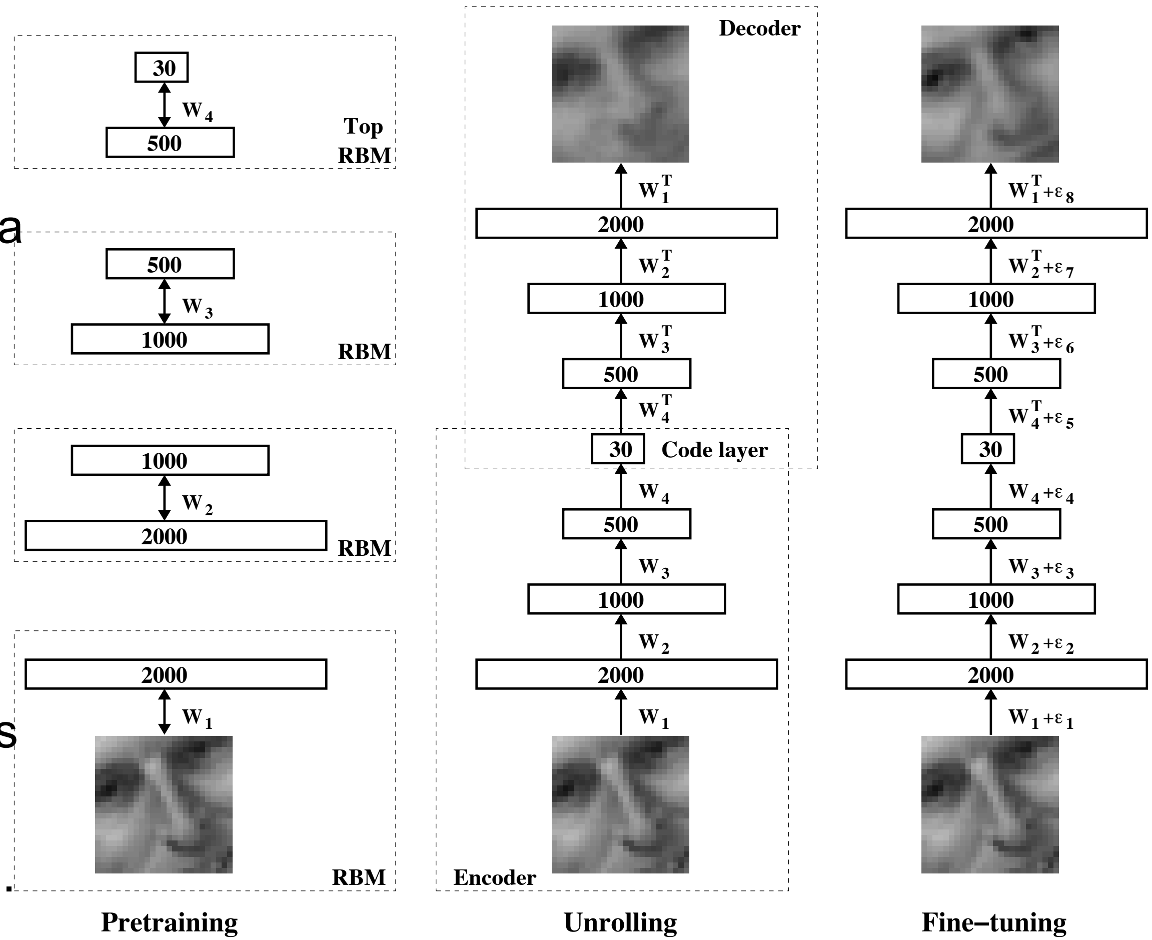
Deep Autoencoder

- Pre-training can be used to initialize a deep autoencoder

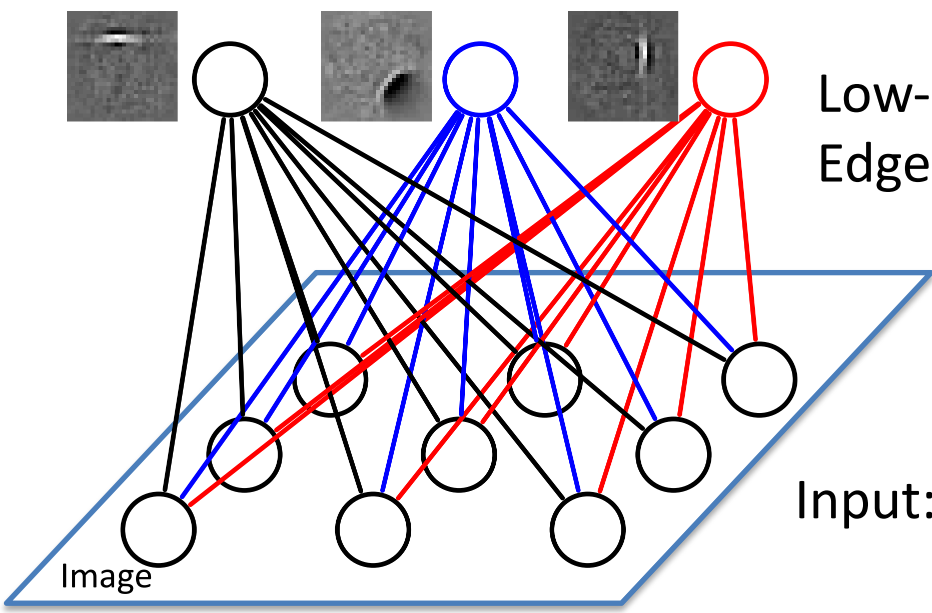
➢ Pre-training initializes the optimization problem in a region with better local optima of the training objective

➢ Each RBM used to initialize parameters both in encoder and decoder (“unrolling”)

➢ Better optimization algorithms can also help: Deep learning via Hessian-free optimization. Martens, 2010



Deep Belief Network



Low-level features: Edges

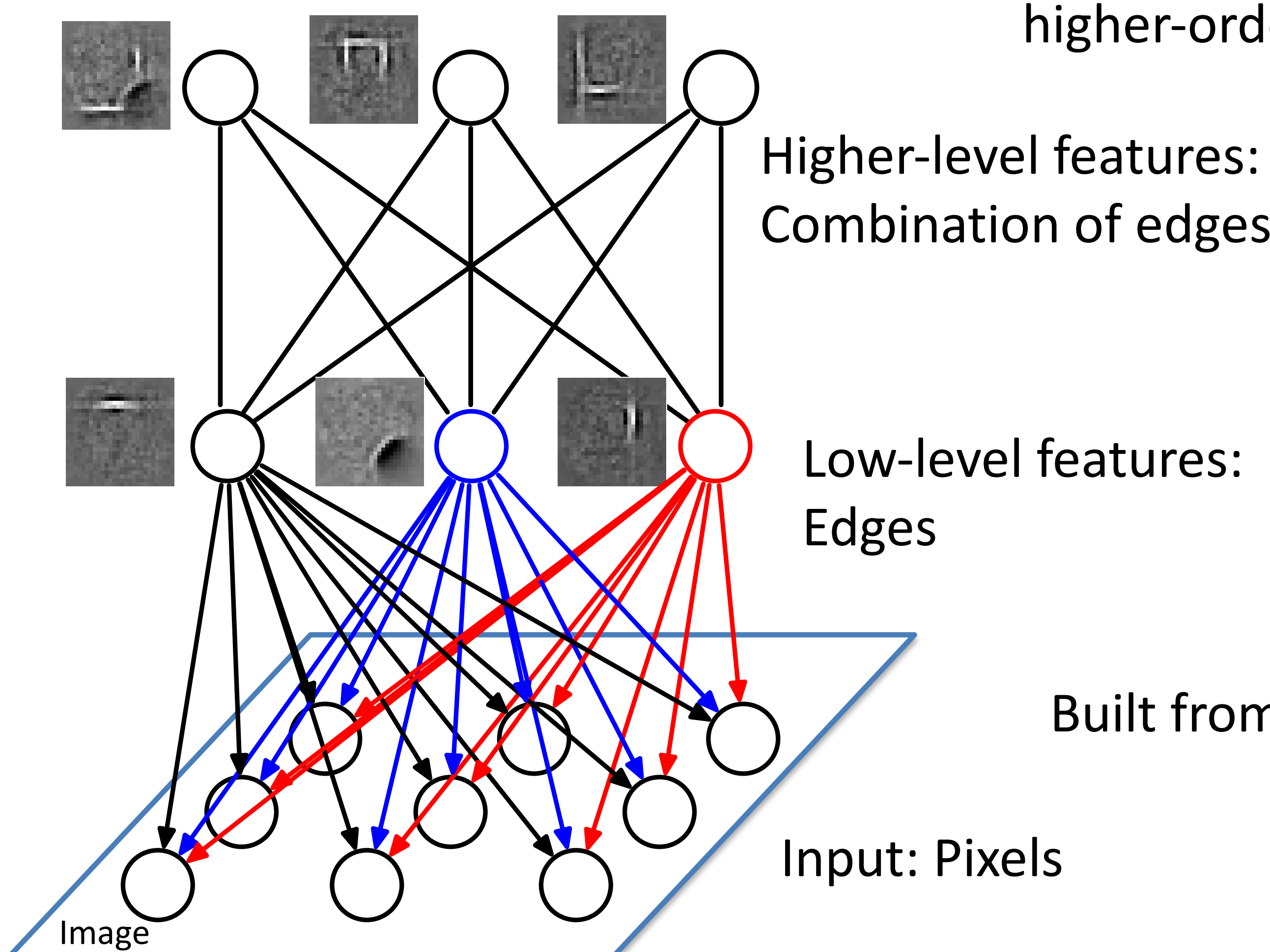
Built from **unlabeled** inputs.

Input: Pixels

(Hinton et.al. Neural Computation 2006)

Deep Belief Network

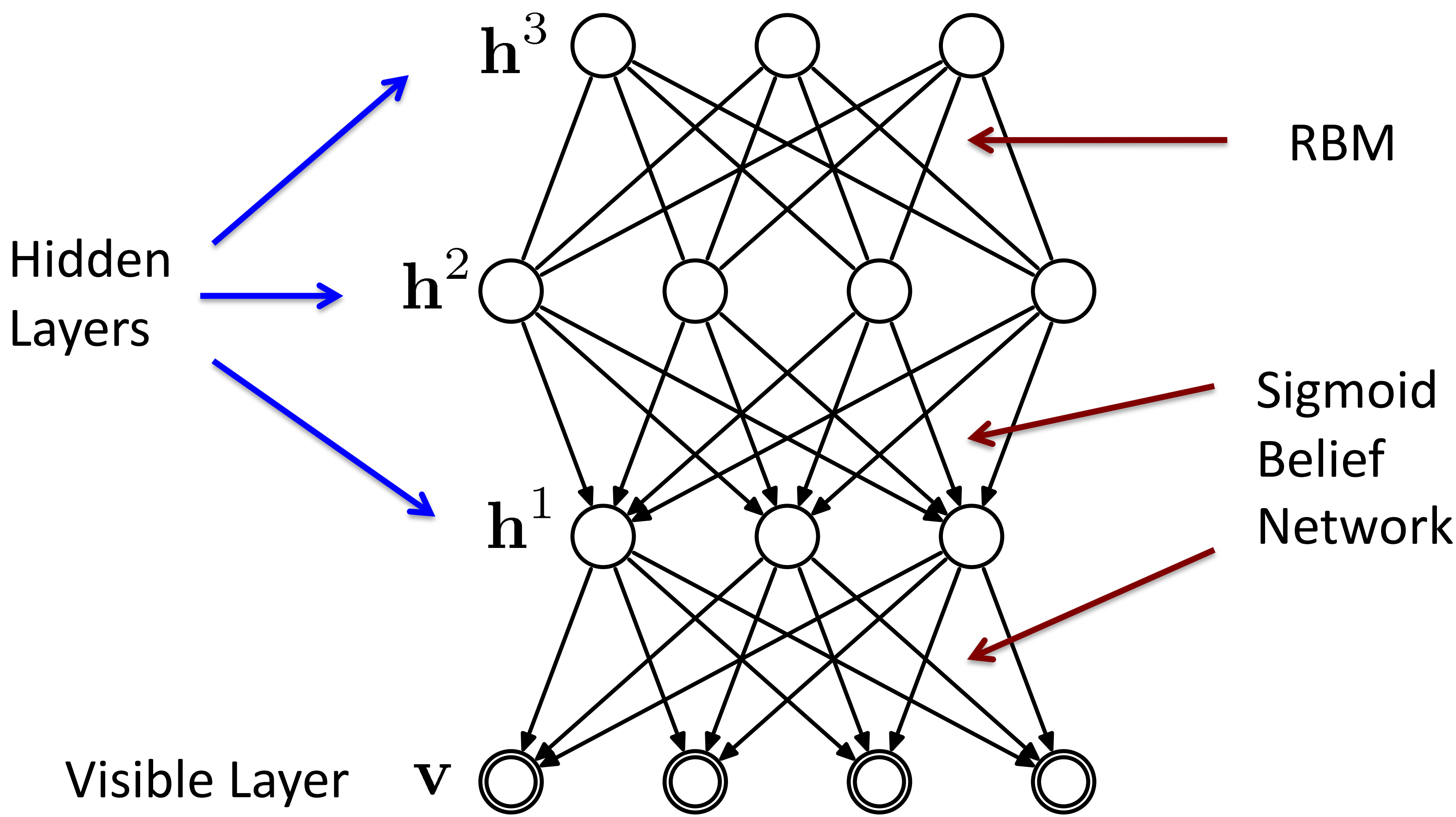
Internal representations capture higher-order statistical structure



Built from **unlabeled** inputs.

(Hinton et.al. Neural Computation 2006)

Deep Belief Network



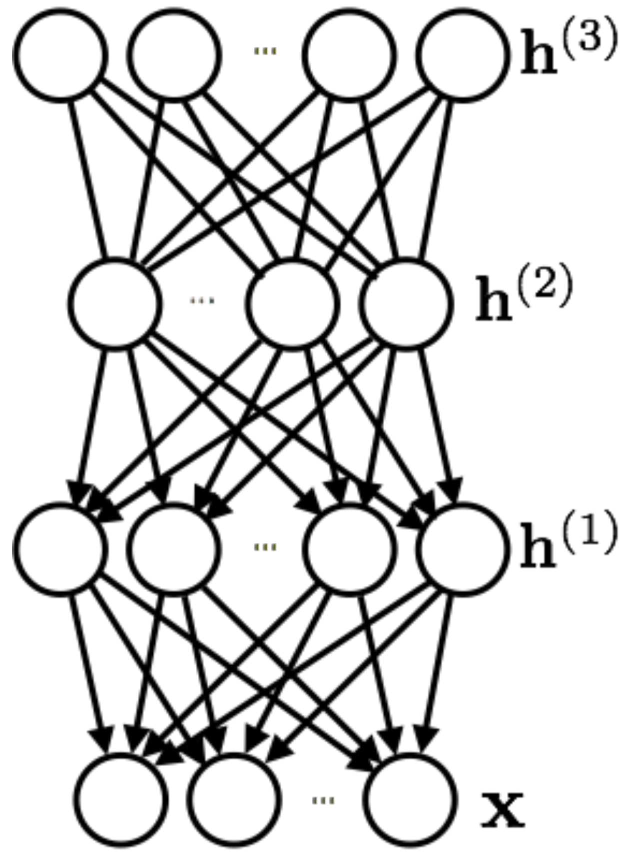
Deep Belief Network

- Deep Belief Networks:
  - it is a generative model that mixes undirected and directed connections between variables
  - top 2 layers' distribution $p(\mathbf{h}^{(2)}, \mathbf{h}^{(3)})$ is an RBM!
  - other layers form a Bayesian network with conditional distributions:

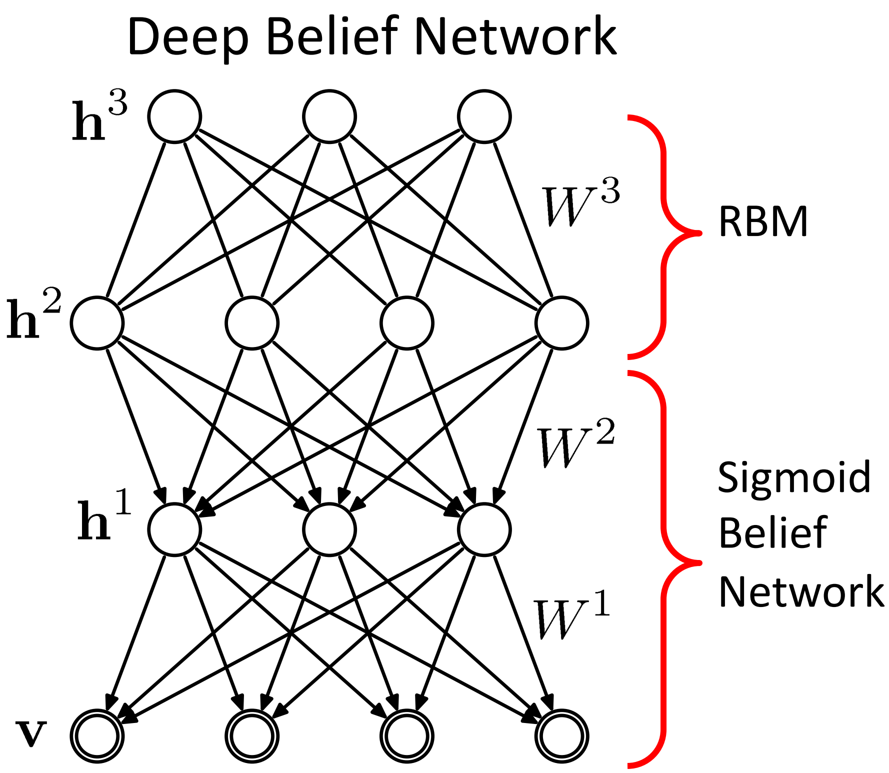
$$p(h_j^{(1)} = 1|\mathbf{h}^{(2)}) = \text{sigm}(\mathbf{b}^{(1)} + \mathbf{W}^{(2)}{}^\top \mathbf{h}^{(2)})$$

$$p(x_i = 1|\mathbf{h}^{(1)}) = \text{sigm}(\mathbf{b}^{(0)} + \mathbf{W}^{(1)}{}^\top \mathbf{h}^{(1)})$$

  - This is not a feed-forward neural network



Deep Belief Network



- top 2 layers' distribution $p(\mathbf{h}^{(2)}, \mathbf{h}^{(3)})$ is an RBM
- other layers form a Bayesian network with conditional distributions:

$$p(h_j^{(1)} = 1|\mathbf{h}^{(2)}) = \text{sigm}(\mathbf{b}^{(1)} + \mathbf{W}^{(2)^\top}\mathbf{h}^{(2)})$$

$$p(x_i = 1|\mathbf{h}^{(1)}) = \text{sigm}(\mathbf{b}^{(0)} + \mathbf{W}^{(1)^\top}\mathbf{h}^{(1)})$$

Deep Belief Network

- The joint distribution of a DBN is as follows

$$p(\mathbf{x}, \mathbf{h}^{(1)}, \mathbf{h}^{(2)}, \mathbf{h}^{(3)}) = p(\mathbf{h}^{(2)}, \mathbf{h}^{(3)})\; p(\mathbf{h}^{(1)} | \mathbf{h}^{(2)})\; p(\mathbf{x} | \mathbf{h}^{(1)})$$

where

$$p(\mathbf{h}^{(2)}, \mathbf{h}^{(3)}) = \exp\left(\mathbf{h}^{(2)}{}^\top \mathbf{W}^{(3)} \mathbf{h}^{(3)} + \mathbf{b}^{(2)}{}^\top \mathbf{h}^{(2)} + \mathbf{b}^{(3)}{}^\top \mathbf{h}^{(3)} \right) / Z$$

$$p(\mathbf{h}^{(1)} | \mathbf{h}^{(2)}) = \prod_j p(h_j^{(1)} | \mathbf{h}^{(2)})$$

$$p(\mathbf{x} | \mathbf{h}^{(1)}) = \prod_i p(x_i | \mathbf{h}^{(1)})$$

- As in a deep feed-forward network, training a DBN is hard

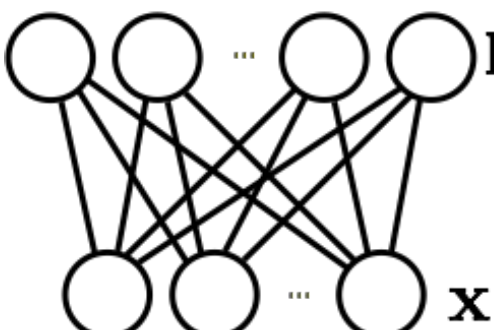
Layer-wise Pretraining

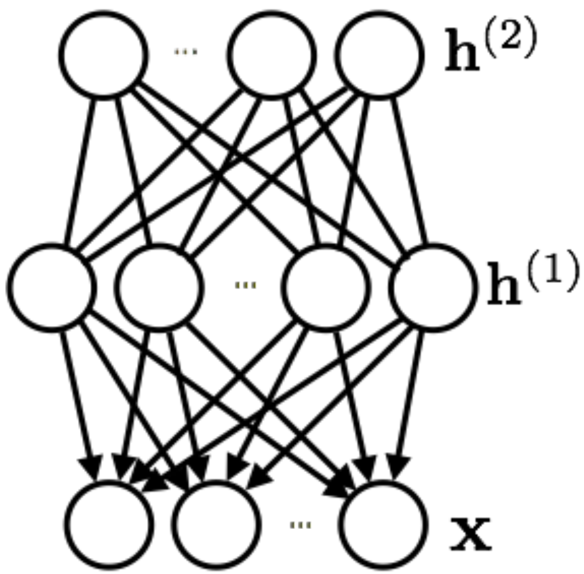
- This is where the RBM stacking procedure comes from:
  - idea: improve prior on last layer by adding another hidden layer

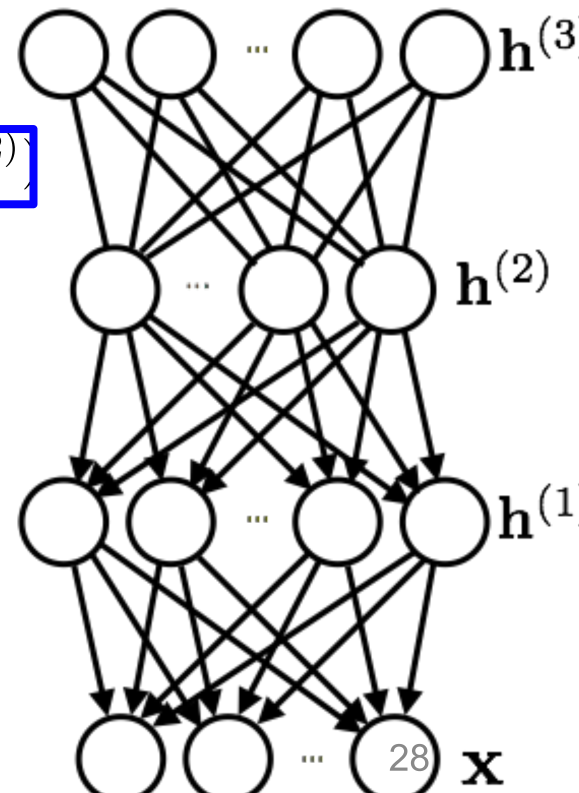
$$p(\mathbf{h}^{(1)}, \mathbf{h}^{(2)}) = p(\mathbf{h}^{(1)}|\mathbf{h}^{(2)}) \sum_{\mathbf{h}^{(3)}} p(\mathbf{h}^{(2)}, \mathbf{h}^{(3)})$$

$$p(\mathbf{x}, \mathbf{h}^{(1)}) = p(\mathbf{x}|\mathbf{h}^{(1)}) \sum_{\mathbf{h}^{(2)}} p(\mathbf{h}^{(1)}, \mathbf{h}^{(2)})$$

$$p(\mathbf{x}) = \sum_{\mathbf{h}^{(1)}} p(\mathbf{x}, \mathbf{h}^{(1)})$$



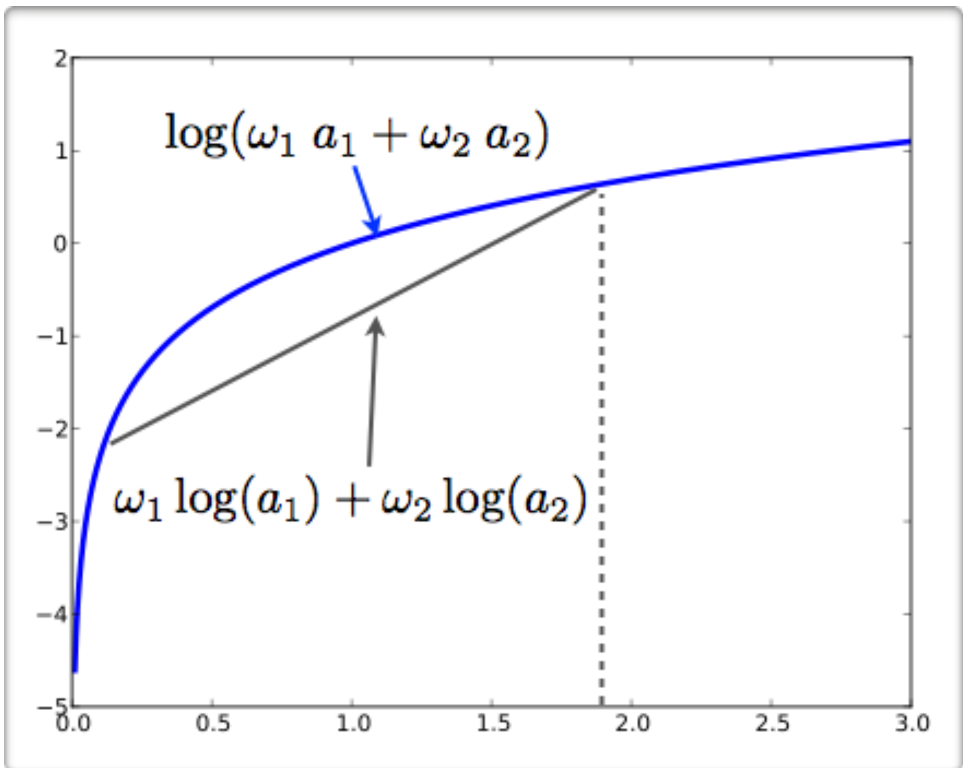




Concavity

$$\log(\sum_i \omega_i \, a_i) \geq \sum_i \omega_i \log(a_i)$$

(where $\sum_i \omega_i = 1$ and $\omega_i \geq 0$)



Variational Bound

- For any model $p(\mathbf{x}, \mathbf{h}^{(1)})$ with latent variables $\mathbf{h}^{(1)}$ we can write:

$$
\begin{aligned}
\log p(\mathbf{x}) &= \log \left(\sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \frac{p(\mathbf{x}, \mathbf{h}^{(1)})}{q(\mathbf{h}^{(1)}|\mathbf{x})} \right) \\
&\geq \sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \log \left(\frac{p(\mathbf{x}, \mathbf{h}^{(1)})}{q(\mathbf{h}^{(1)}|\mathbf{x})} \right) \\
&= \sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \log p(\mathbf{x}, \mathbf{h}^{(1)}) \\
&\quad - \sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \log q(\mathbf{h}^{(1)}|\mathbf{x})
\end{aligned}
$$

where $q(\mathbf{h}^{(1)}|\mathbf{x})$ is any approximation to $p(\mathbf{h}^{(1)}|\mathbf{x})$

Variational Bound

- This is called a variational bound

$$\log p(\mathbf{x}) \geq \sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \log p(\mathbf{x}, \mathbf{h}^{(1)}) - \sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \log q(\mathbf{h}^{(1)}|\mathbf{x})$$

  - if $q(\mathbf{h}^{(1)}|\mathbf{x})$ is equal to the true conditional $p(\mathbf{h}^{(1)}|\mathbf{x})$, then we have an equality – the bound is tight!
  - the more $q(\mathbf{h}^{(1)}|\mathbf{x})$ is different from $p(\mathbf{h}^{(1)}|\mathbf{x})$ the less tight the bound is.

Variational Bound

- This is called a variational bound

$$\log p(\mathbf{x}) \geq \sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \log p(\mathbf{x}, \mathbf{h}^{(1)}) - \sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \log q(\mathbf{h}^{(1)}|\mathbf{x})$$

➢ In fact, difference between the left and right terms is the KL divergence between $q(\mathbf{h}^{(1)}|\mathbf{x})$ and $p(\mathbf{h}^{(1)}|\mathbf{x})$:

$$\mathrm{KL}(q||p) = \sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \log \left(\frac{q(\mathbf{h}^{(1)}|\mathbf{x})}{p(\mathbf{h}^{(1)}|\mathbf{x})} \right)$$

Variational Bound

- This is called a variational bound

$$\log p(\mathbf{x}) \quad \geq \quad \sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \left(\log p(\mathbf{x}|\mathbf{h}^{(1)}) + \log p(\mathbf{h}^{(1)})\right)$$
$$- \sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \log q(\mathbf{h}^{(1)}|\mathbf{x})$$

- ➢ for a single hidden layer DBN (i.e. an RBM), both the likelihood $p(\mathbf{x}|\mathbf{h}^{(1)})$ and the prior $p(\mathbf{h}^{(1)})$ depend on the parameters of the first layer.

- ➢ we can now improve the model by building a better prior $p(\mathbf{h}^{(1)})$

Variational Bound

adding 2nd layer means untying the parameters

- This is called a variational bound

$$\log p(\mathbf{x}) \geq \sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \left(\log p(\mathbf{x}|\mathbf{h}^{(1)}) + \log p(\mathbf{h}^{(1)}) \right) - \sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \log q(\mathbf{h}^{(1)}|\mathbf{x})$$

- When adding a second layer, we model $p(\mathbf{h}^{(1)})$ using a separate set of parameters
    - ➢ they are the parameters of the RBM involving $\mathbf{h}^{(1)}$ and $\mathbf{h}^{(2)}$
    - ➢ $p(\mathbf{h}^{(1)})$ is now the marginalization of the second hidden layer

$$p(\mathbf{h}^{(1)}) = \sum_{\mathbf{h}^{(2)}} p(\mathbf{h}^{(1)}, \mathbf{h}^{(2)})$$

Variational Bound

adding 2nd layer means
untying the parameters

- This is called a variational bound

$$\log p(\mathbf{x}) \geq \sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \left(\log p(\mathbf{x}|\mathbf{h}^{(1)}) + \log p(\mathbf{h}^{(1)}) \right)$$

$$- \sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \log q(\mathbf{h}^{(1)}|\mathbf{x})$$

➢ we can train the parameters c g
the bound. This is equivalent
other terms are constant:

$$- \sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \log p(\mathbf{h}^{(1)})$$

Layerwise pretraining improves variational lower bound

➢ this is like training an RBM on data generated from $q(\mathbf{h}^{(1)}|\mathbf{x})$!

Variational Bound

- This is called a variational bound

adding 2nd layer means untying the parameters

$$\log p(\mathbf{x}) \geq \sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \left(\log p(\mathbf{x}|\mathbf{h}^{(1)}) + \log p(\mathbf{h}^{(1)})\right) - \sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \log q(\mathbf{h}^{(1)}|\mathbf{x})$$

- for $q(\mathbf{h}^{(1)}|\mathbf{x})$ we use the posterior of the first layer RBM. This is equivalent to a feed-forward (sigmoidal) layer, followed by sampling
- by initializing the weights of the second layer RBM as the transpose of the first layer weights, the bound is initially tight!
- a 2-layer DBN with tied weights is equivalent to a 1-layer RBM

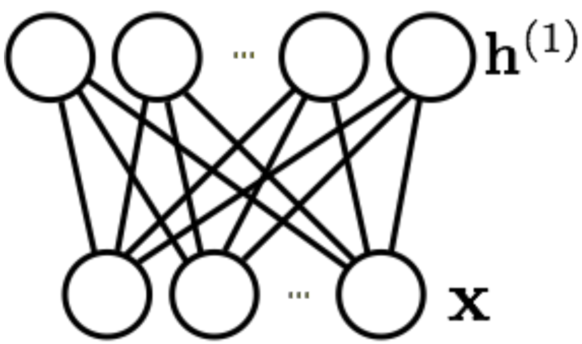
Layer-wise Pretraining

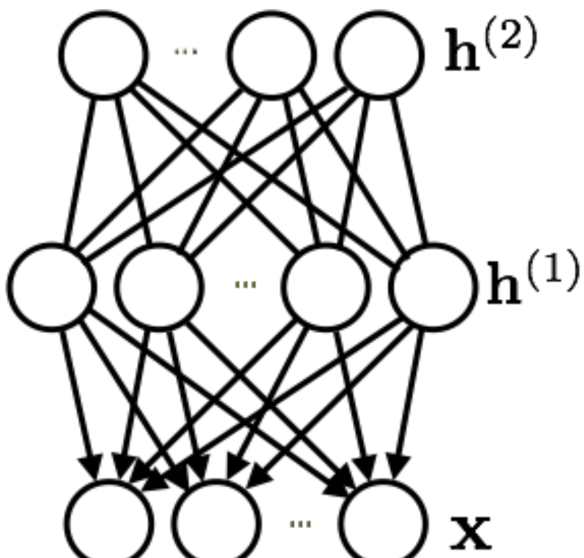
- This is where the RBM stacking procedure comes from:
  - ➢ idea: improve prior on last layer by adding another hidden layer

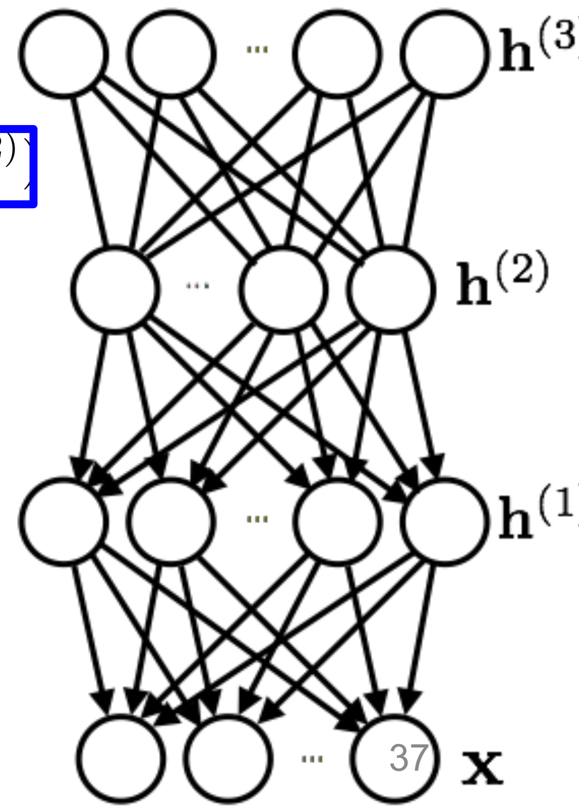
$$p(\mathbf{h}^{(1)}, \mathbf{h}^{(2)}) = p(\mathbf{h}^{(1)}|\mathbf{h}^{(2)}) \sum_{\mathbf{h}^{(3)}} p(\mathbf{h}^{(2)}, \mathbf{h}^{(3)})$$

$$p(\mathbf{x}, \mathbf{h}^{(1)}) = p(\mathbf{x}|\mathbf{h}^{(1)}) \sum_{\mathbf{h}^{(2)}} p(\mathbf{h}^{(1)}, \mathbf{h}^{(2)})$$

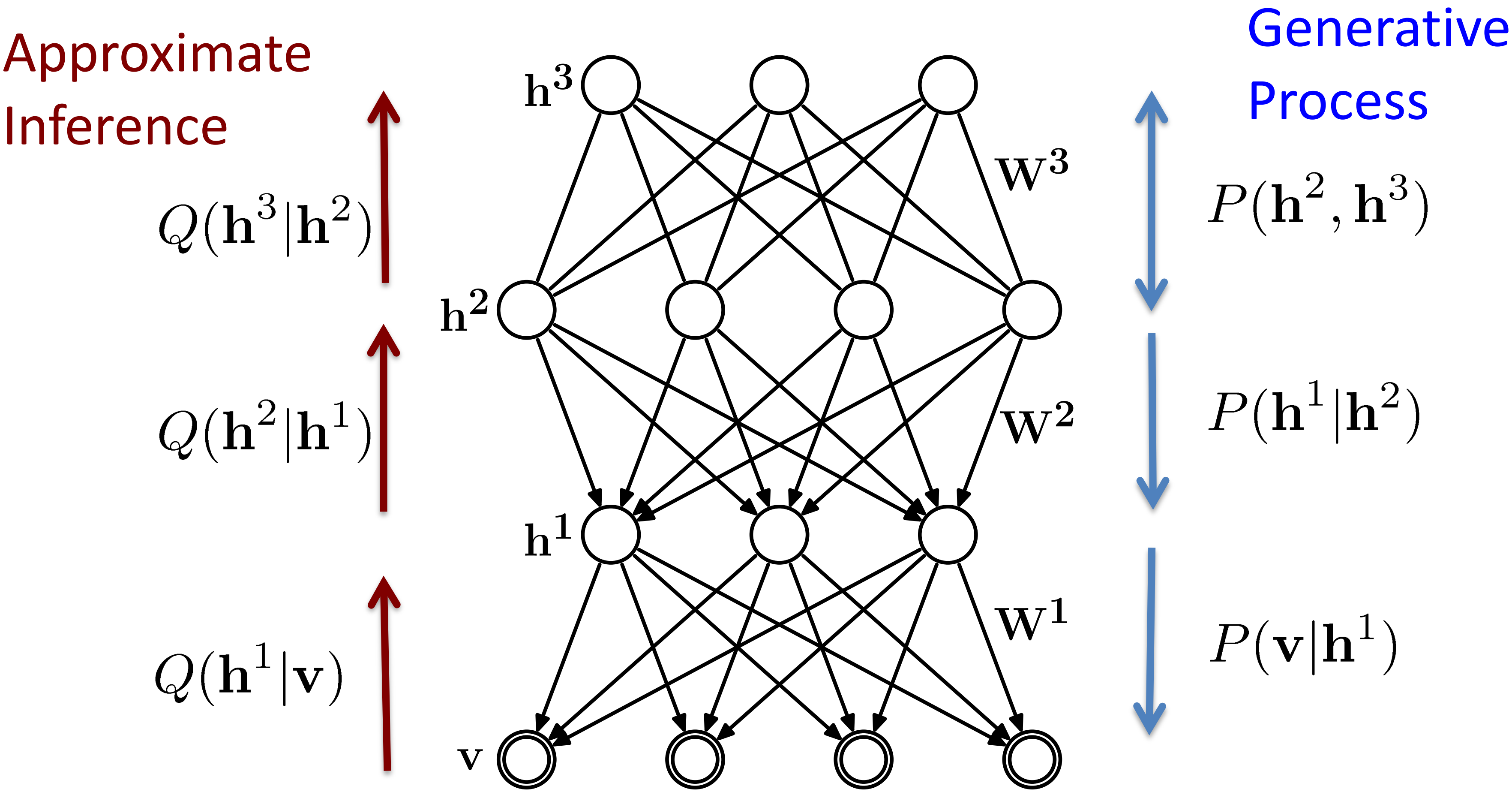
$$p(\mathbf{x}) = \sum_{\mathbf{h}^{(1)}} p(\mathbf{x}, \mathbf{h}^{(1)})$$







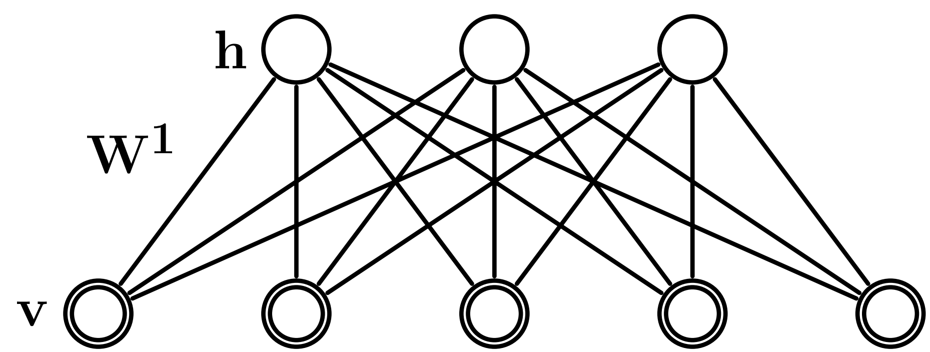
Deep Belief Network



$$Q(\mathbf{h}^t|\mathbf{h}^{t-1}) = \prod_j \sigma\left(\sum_i W^t h_i^{t-1}\right) \qquad P(\mathbf{h}^{t-1}|\mathbf{h}^t) = \prod_j \sigma\left(\sum_i W^t h_i^t\right)$$

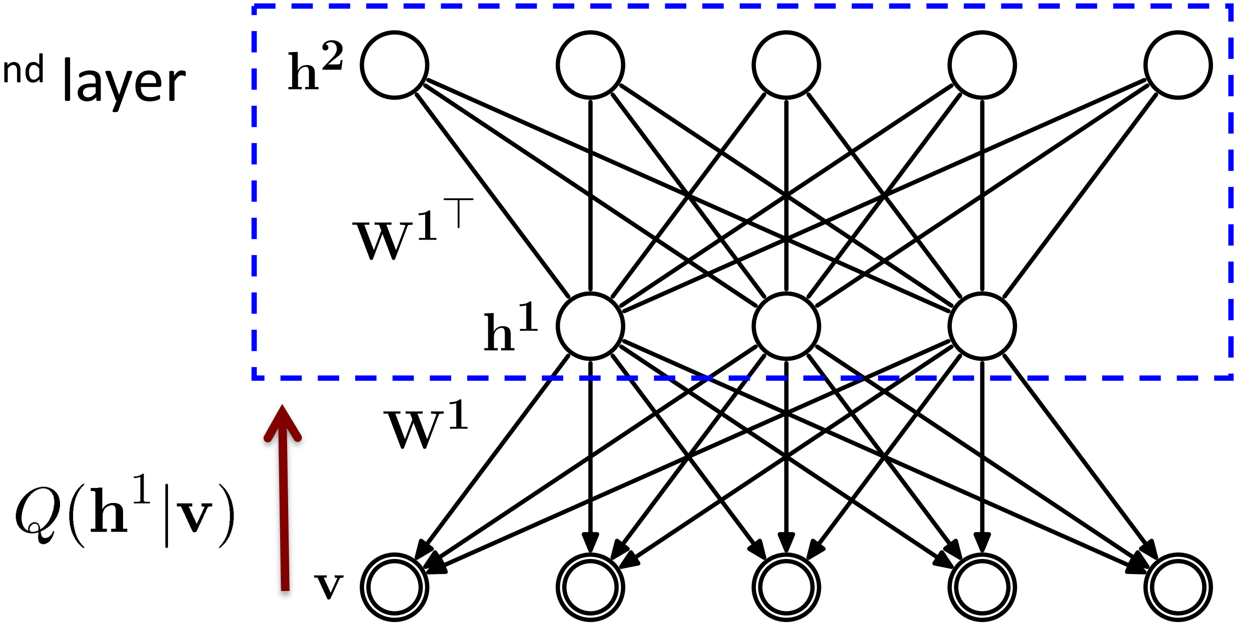
DBN Layer-wise Training

- Learn an RBM with an input layer v=x and a hidden layer h.



DBN Layer-wise Training

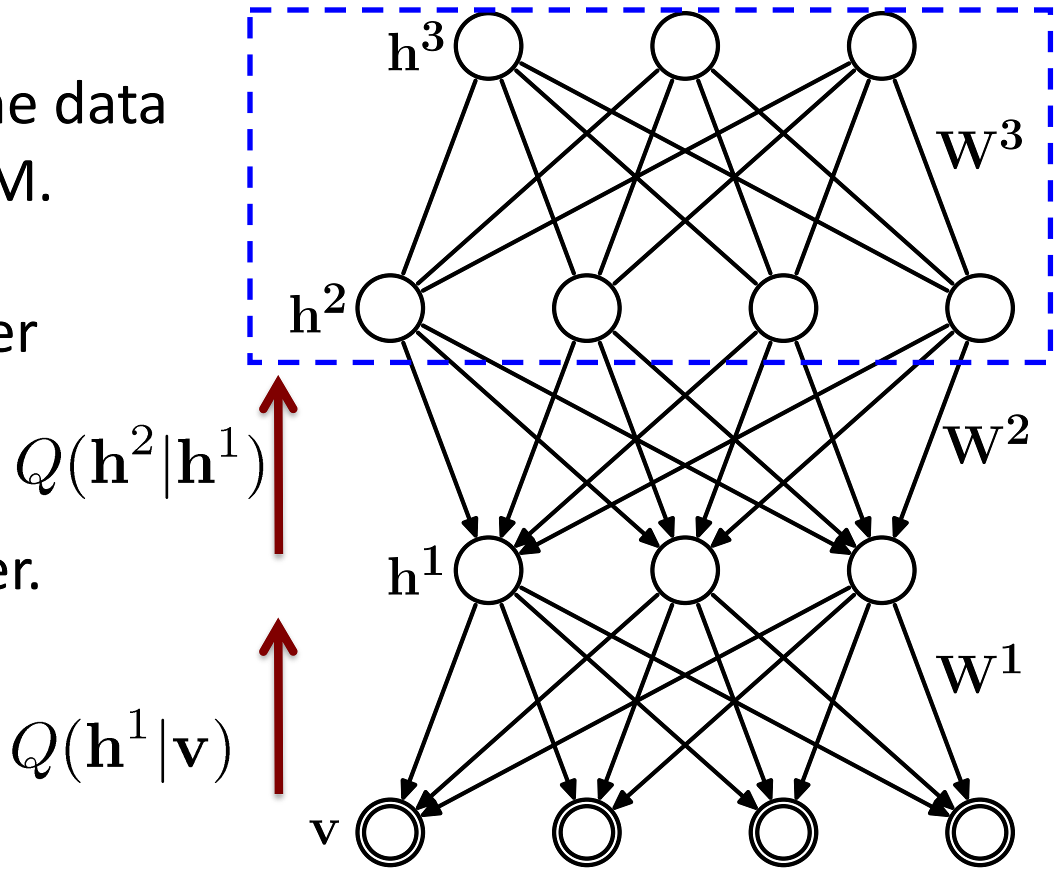
- Learn an RBM with an input layer v=x and a hidden layer h.
- Treat inferred values $Q(\mathbf{h}^1|\mathbf{v}) = P(\mathbf{h}^1|\mathbf{v})$ as the data for training 2nd-layer RBM.
- Learn and freeze 2nd layer RBM.



DBN Layer-wise Training

- Learn an RBM with an input layer v=x and a hidden layer h.
- Treat inferred values $Q(\mathbf{h}^1|\mathbf{v}) = P(\mathbf{h}^1|\mathbf{v})$ as the data for training 2nd-layer RBM.
- Learn and freeze 2nd layer RBM.
- Proceed to the next layer.

Unsupervised Feature Learning.

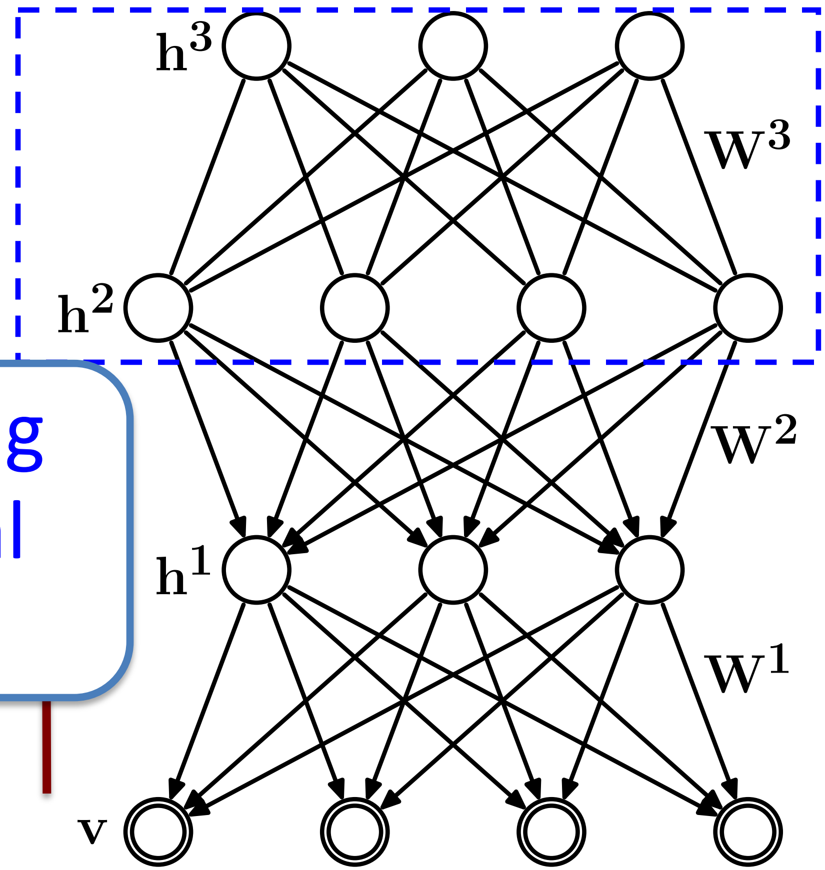


DBN Layer-wise Training

- Learn an RBM with an input layer v=x and a hidden layer h.
- Treat inferred values $Q(\mathbf{h}^1|\mathbf{v}) = P(\mathbf{h}^1|\mathbf{v})$ as the data for training 2nd-layer RBM.
- Learn and freeze 2nd layer RBM
- Proc

Layerwise pretraining improves variational lower bound

Unsupervised Feature Learning.



$Q(\mathbf{h}^1|\mathbf{v})$

Deep Belief Networks

- This process of adding layers can be repeated recursively
  - we obtain the greedy layer-wise pre-training procedure for neural networks
- We now see that this procedure corresponds to maximizing a bound on the likelihood of the data in a DBN
  - in theory, if our approximation $q(\mathbf{h}^{(1)}|\mathbf{x})$ is very far from the true posterior, the bound might be very loose
  - this only means we might not be improving the true likelihood
  - we might still be extracting better features!
- Fine-tuning is done by the Up-Down algorithm
  - A fast learning algorithm for deep belief nets. Hinton, Teh, Osindero, 2006.

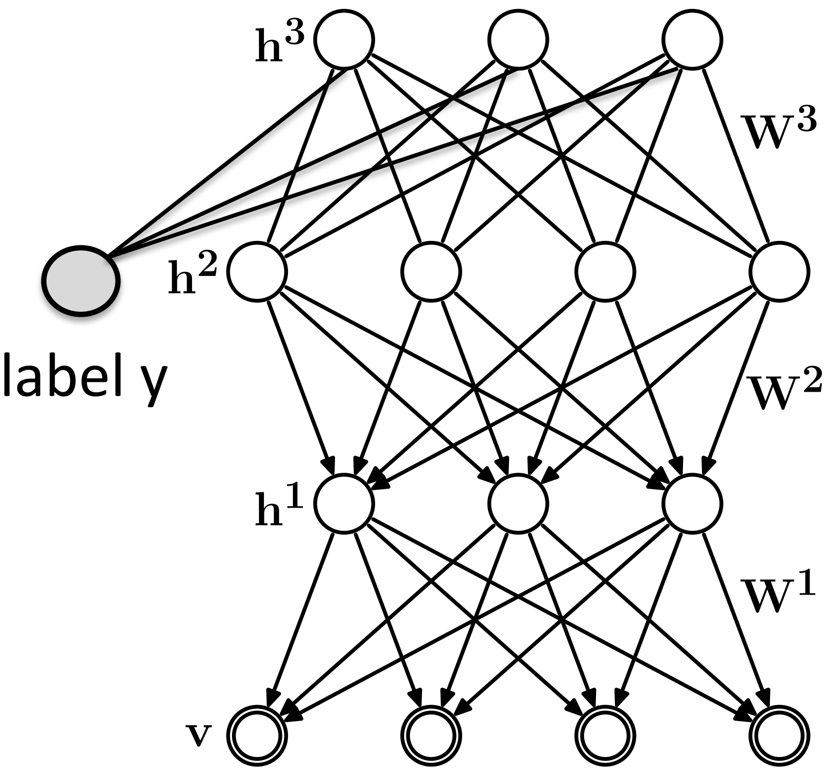
Supervised Learning with DBNs

- If we have access to label information, we can train the joint generative model by maximizing the joint log-likelihood of data and labels

$$\log P(\mathbf{y}, \mathbf{v})$$

- Discriminative fine-tuning:
  - Use DBN to initialize a multilayer neural network.
  - Maximize the conditional distribution:

$$\log P(\mathbf{y}|\mathbf{v})$$

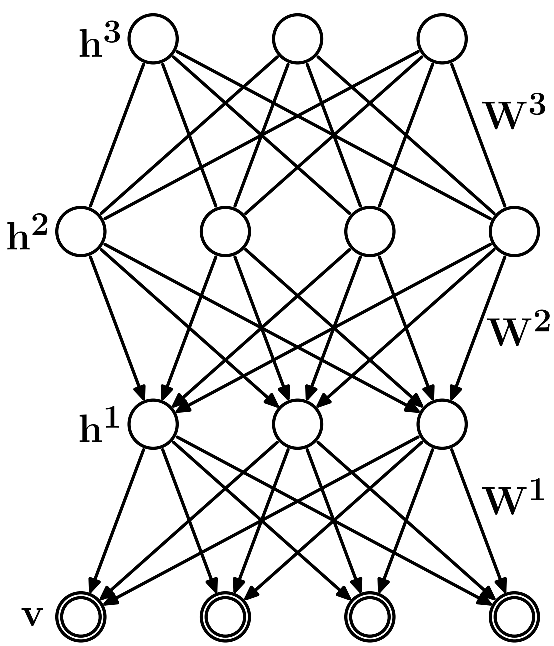


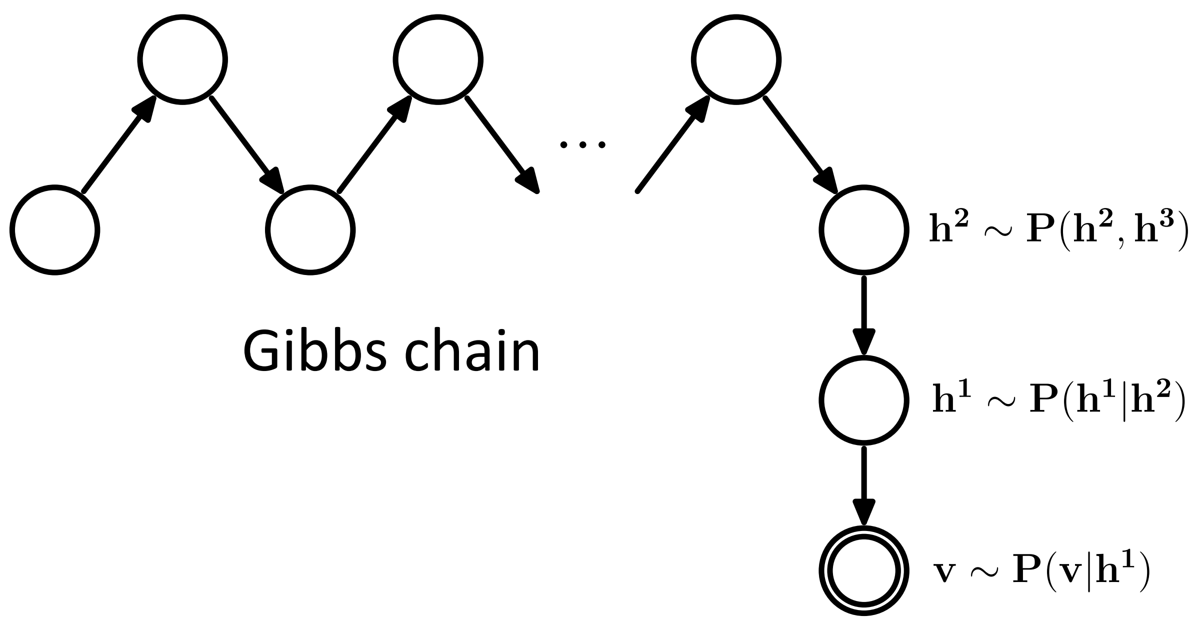
Sampling from DBNs

- To sample from the DBN model:

$$P(\mathbf{v}, \mathbf{h}^1, \mathbf{h}^2, \mathbf{h}^3) = P(\mathbf{v}|\mathbf{h}^1)P(\mathbf{h}^1|\mathbf{h}^2)P(\mathbf{h}^2, \mathbf{h}^3)$$

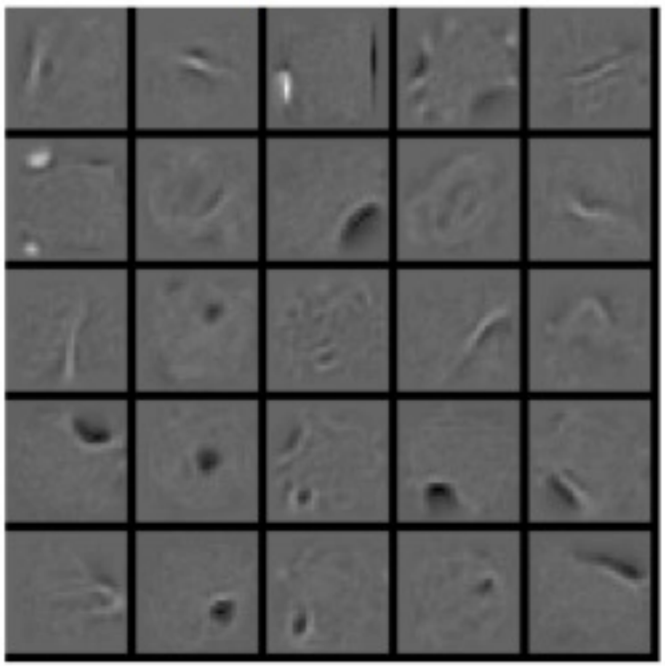
- Sample h^2 using alternating Gibbs sampling from RBM.
- Sample lower layers using sigmoid belief network.



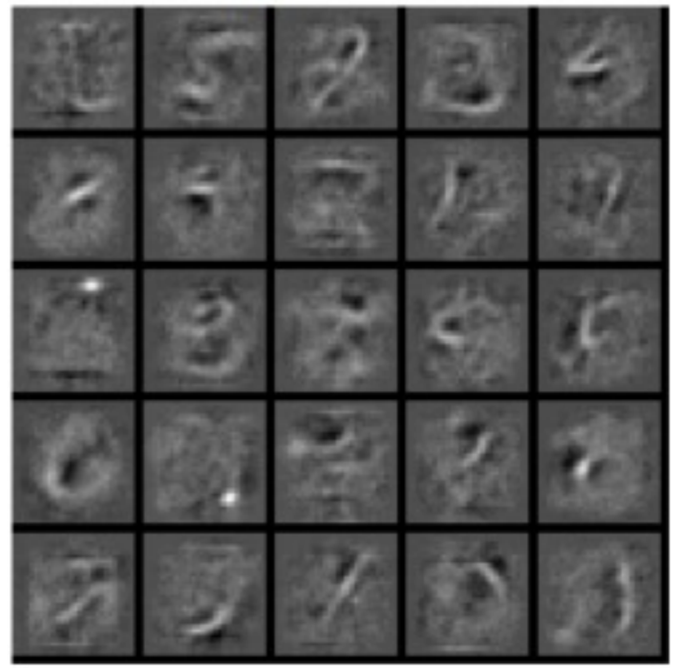


Learned Features

1^{st}-layer features

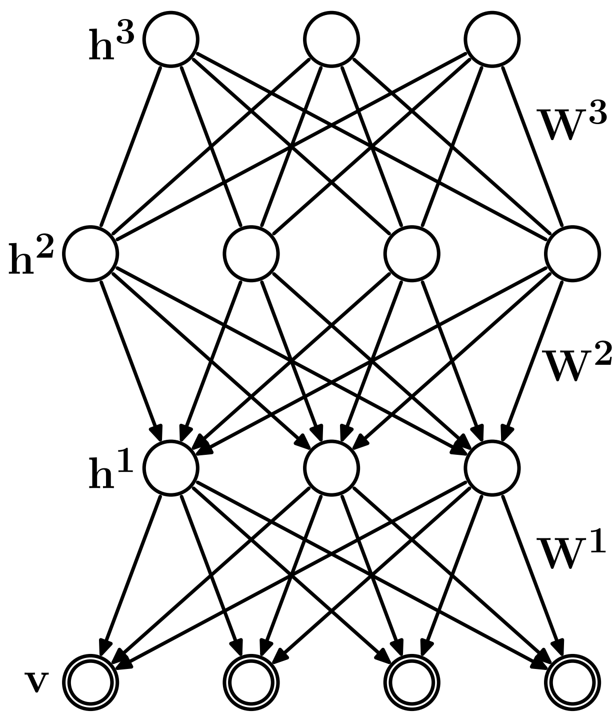


2^{nd}-layer features

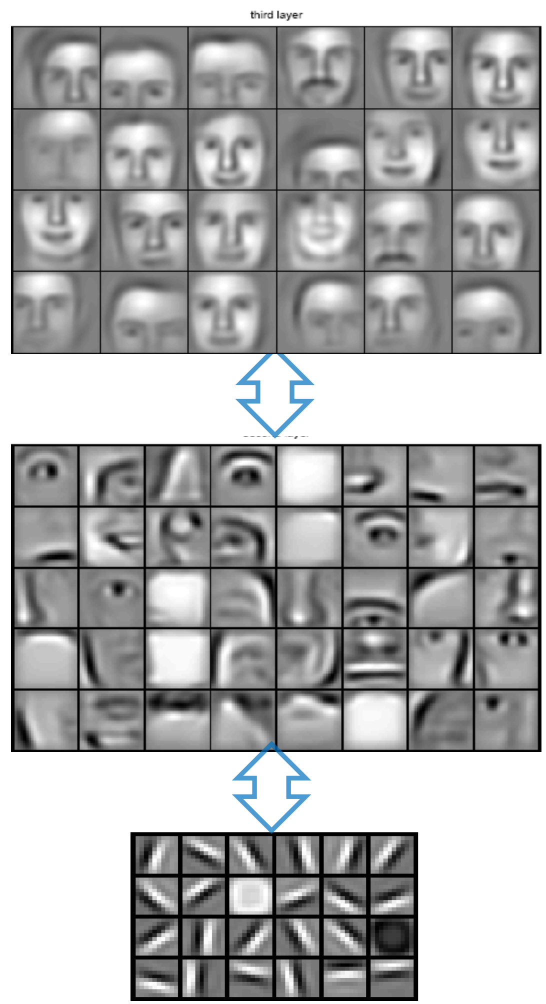


Learning Part-based Representation

Convolutional DBN



Faces



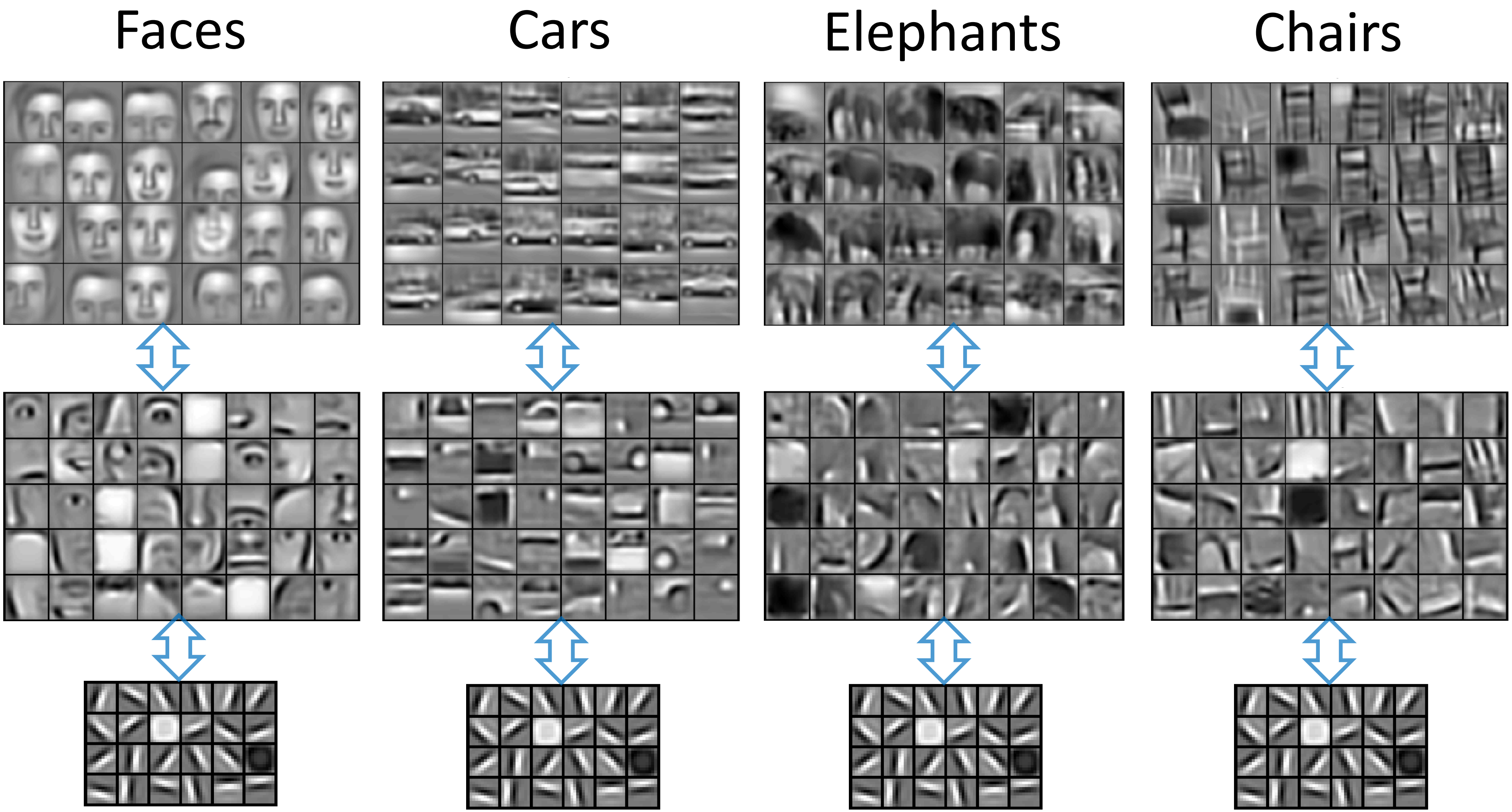
Groups of parts.

Object Parts

Trained on face images.

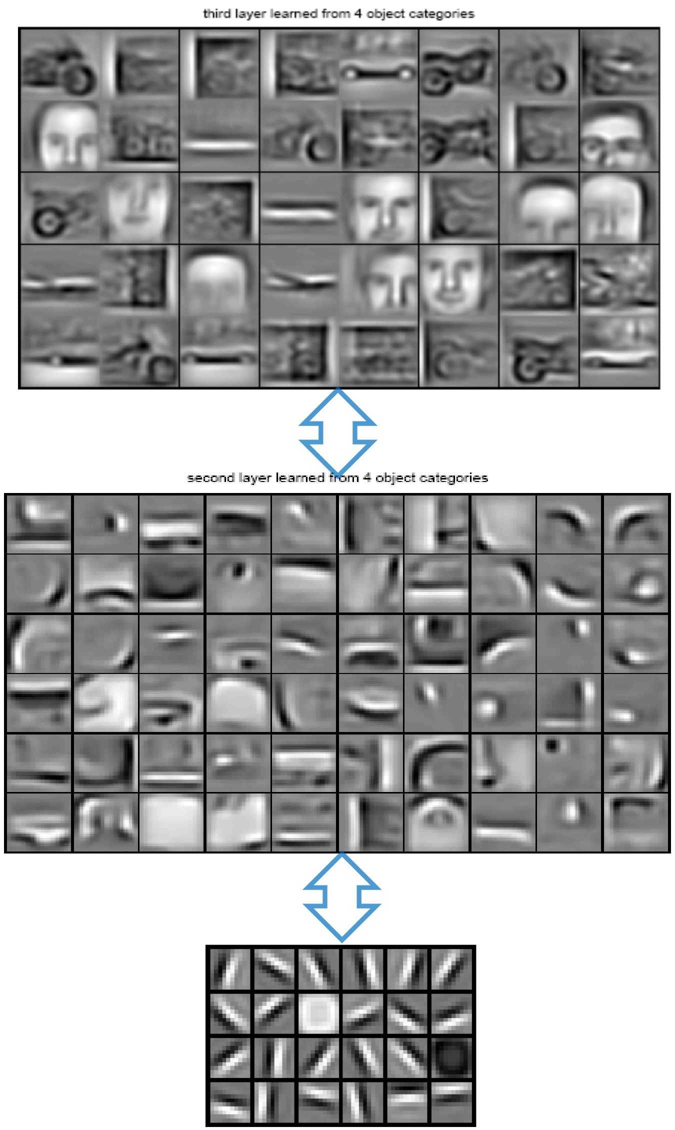
Lee et.al., ICML 2009

Learning Part-based Representation



Lee et.al., ICML 2009

Learning Part-based Representation



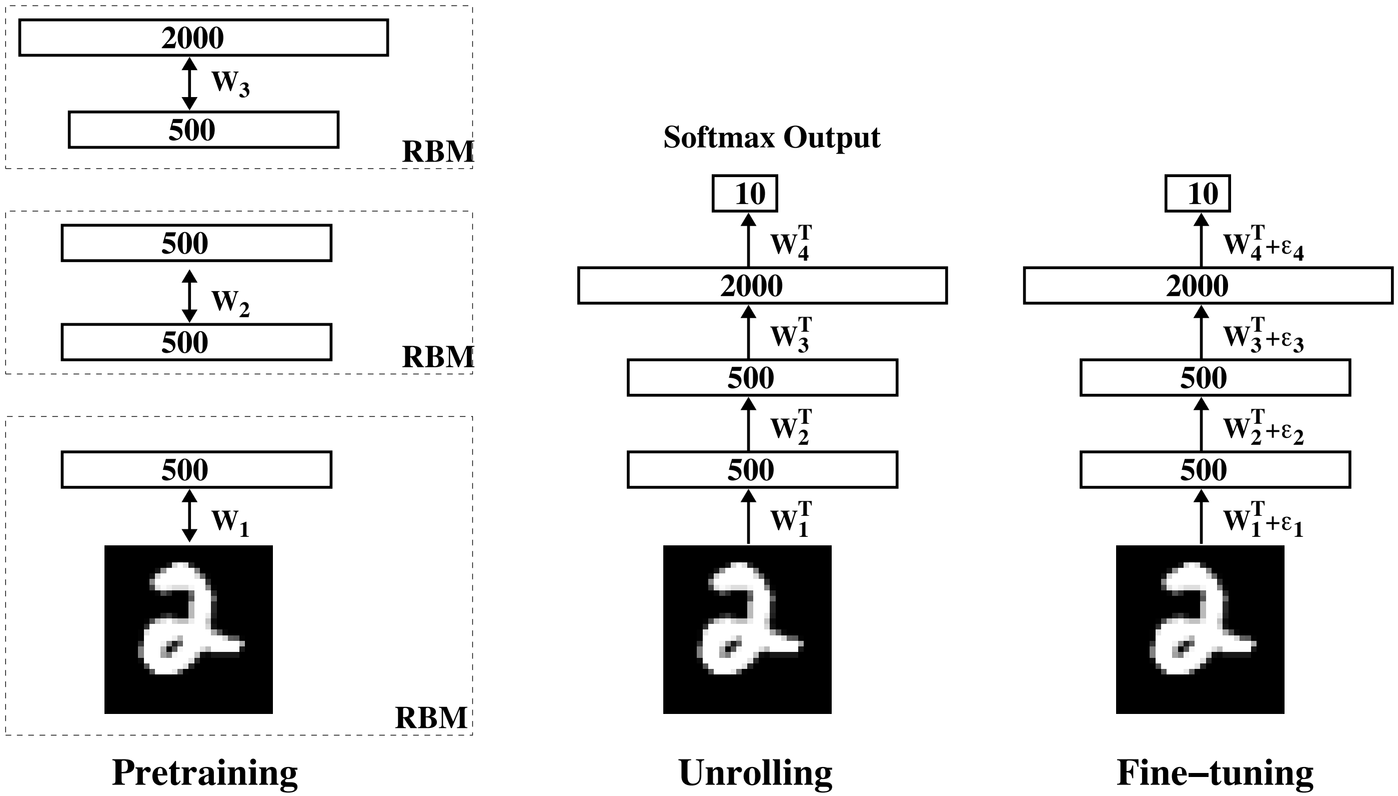
Groups of parts.

Class-specific object parts

Trained from multiple classes (cars, faces, motorbikes, airplanes).

Lee et.al., ICML 2009

DBNs for Classification



- After layer-by-layer **unsupervised pretraining**, discriminative fine-tuning by backpropagation achieves an error rate of 1.2% on MNIST. SVM's get 1.4% and randomly initialized backprop gets 1.6%.

- Clearly unsupervised learning helps generalization. It ensures that most of the information in the weights comes from modeling the input data.

(Hinton and Salakhutdinov, Science 2006)

DBNs for Regression

Predicting the orientation of a face patch

Training Data: 1000 face patches of 30 training people.

Test Data: 1000 face patches of **10 new people**.

Regression Task: predict orientation of a new face.

Gaussian Processes with spherical Gaussian kernel achieves a RMSE (root mean squared error) of 16.33 degree.

(Salakhutdinov and Hinton, NIPS 2007)

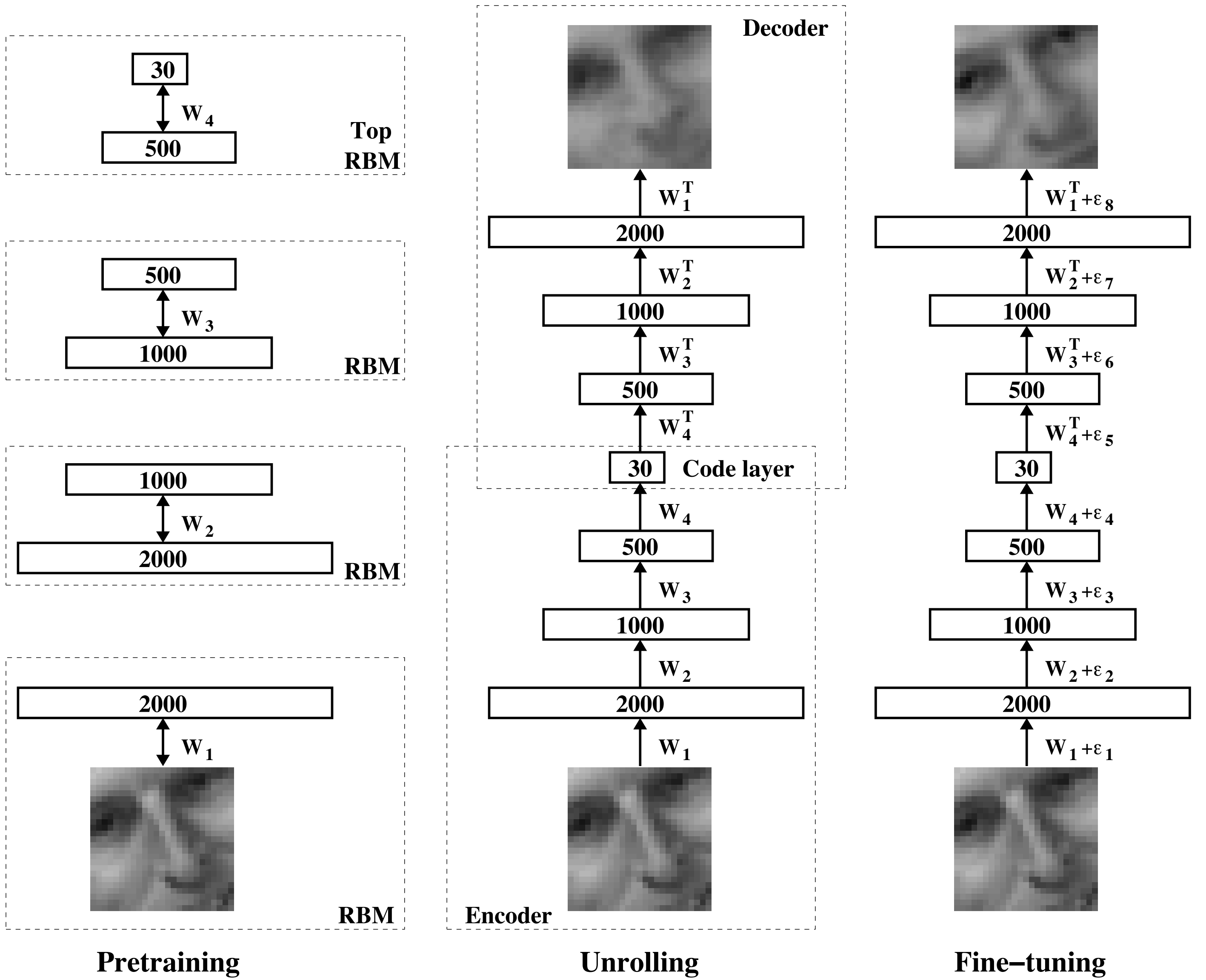
DBNs for Regression

Additional Unlabeled Training Data: 12000 face patches from 30 training people.

- Pretrain a stack of RBMs: 784-1000-1000-1000.
- **Features were extracted with no idea of the final task.**

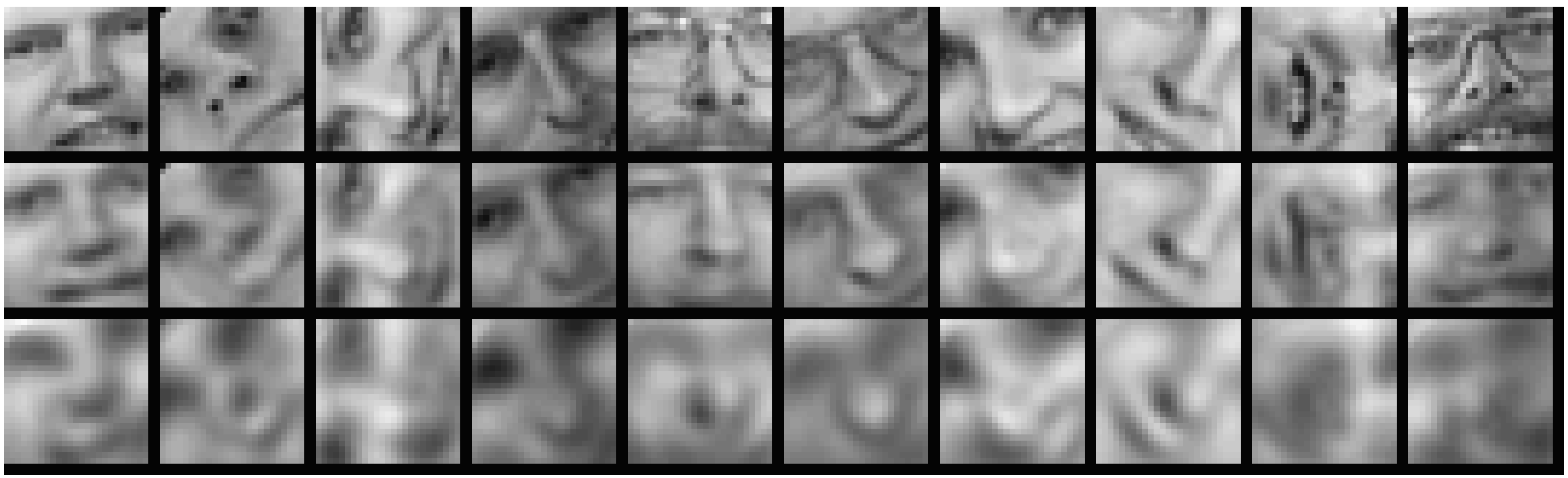
The same GP on the top-level features:	RMSE: 11.22
GP with fine-tuned covariance Gaussian kernel:	RMSE: 6.42
Standard GP without using DBNs:	RMSE: 16.33

Deep Autoencoders



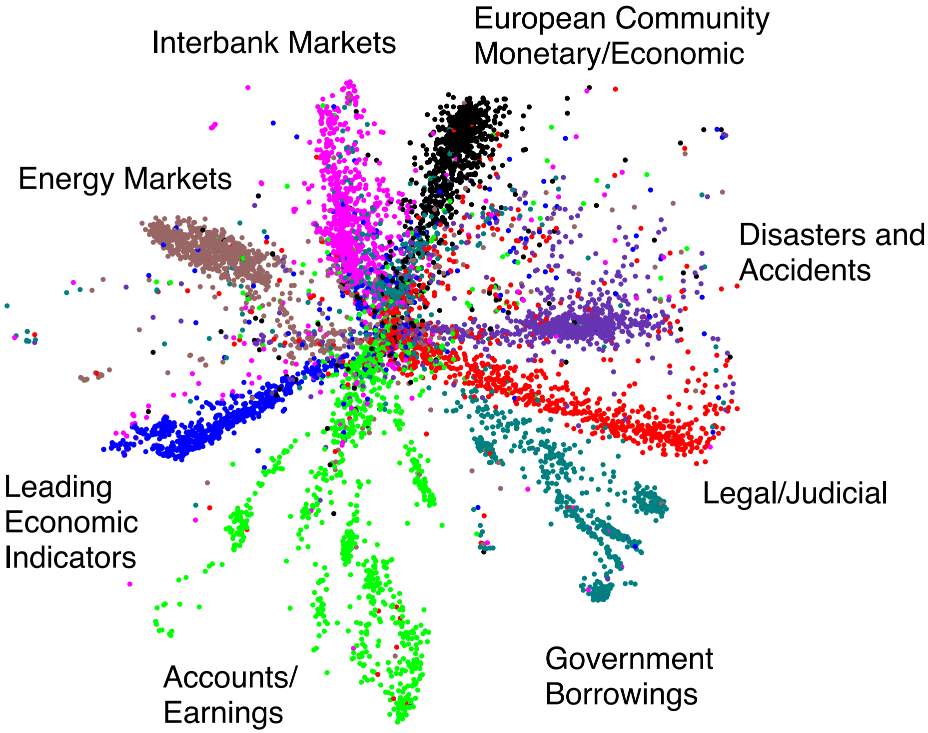
Deep Autoencoders

- We used 25x25 – 2000 – 1000 – 500 – 30 autoencoder to extract 30-D real-valued codes for Olivetti face patches.

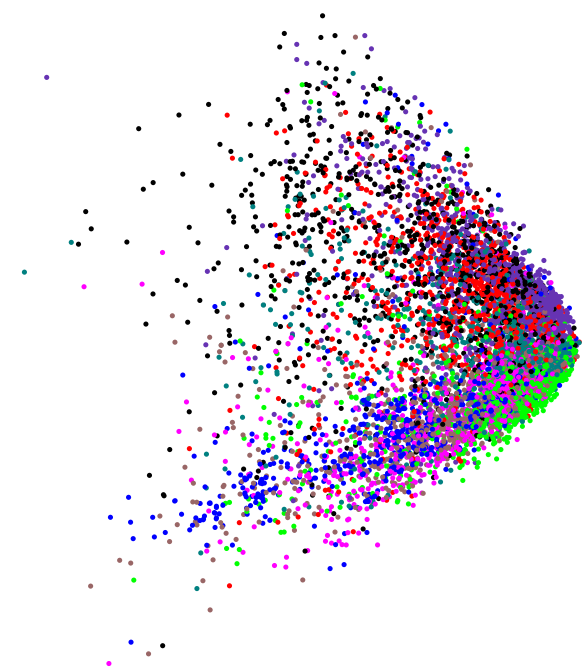


- **Top**: Random samples from the test dataset.
- **Middle**: Reconstructions by the 30-dimensional deep autoencoder.
- **Bottom**: Reconstructions by the 30-dimentinoal PCA.

Information Retrieval



2-D LSA space



- The Reuters Corpus Volume II contains 804,414 newswire stories (randomly split into **402,207 training** and **402,207 test).**

- "Bag-of-words" representation: each article is represented as a vector containing the counts of the most frequently used 2000 words in the training set.

(Hinton and Salakhutdinov, Science 2006)