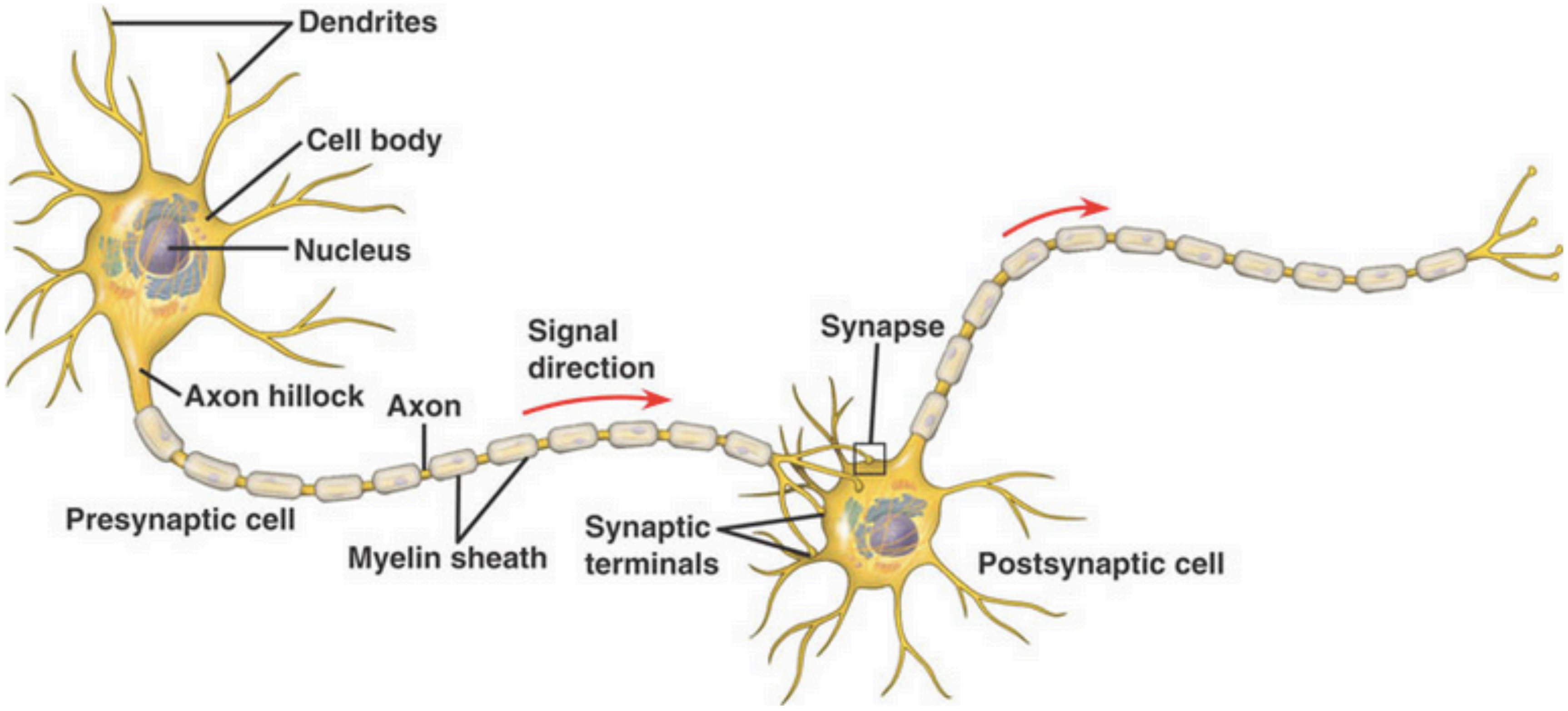
Neural Networks

Machine Learning 10-601**B**

Seyoung Kim

Many of these slides are derived from Tom Mitchell. Thanks!

Neural Networks: Biological Motivation



Logistic Regression is linear classifier

Assumes the following functional form for P(Y|X):

$$P(Y=1|X) = \frac{1}{1+\exp(-(w_0 + \sum_i w_i X_i))}$$

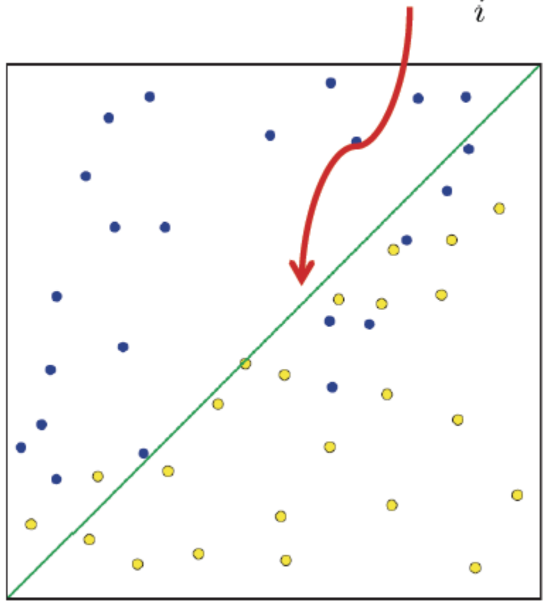
Decision boundary:

$$P(Y=0|X) \overset{0}{\underset{1}{\gtrless}} P(Y=1|X)$$

$$0 \overset{0}{\underset{1}{\gtrless}} w_0 + \sum_i w_i X_i$$

(Linear Decision Boundary)

$$w_0 + \sum_i w_i X_i = 0$$



[slide: Aarti Singh]

Artificial Neural Networks to learn f: X → Y

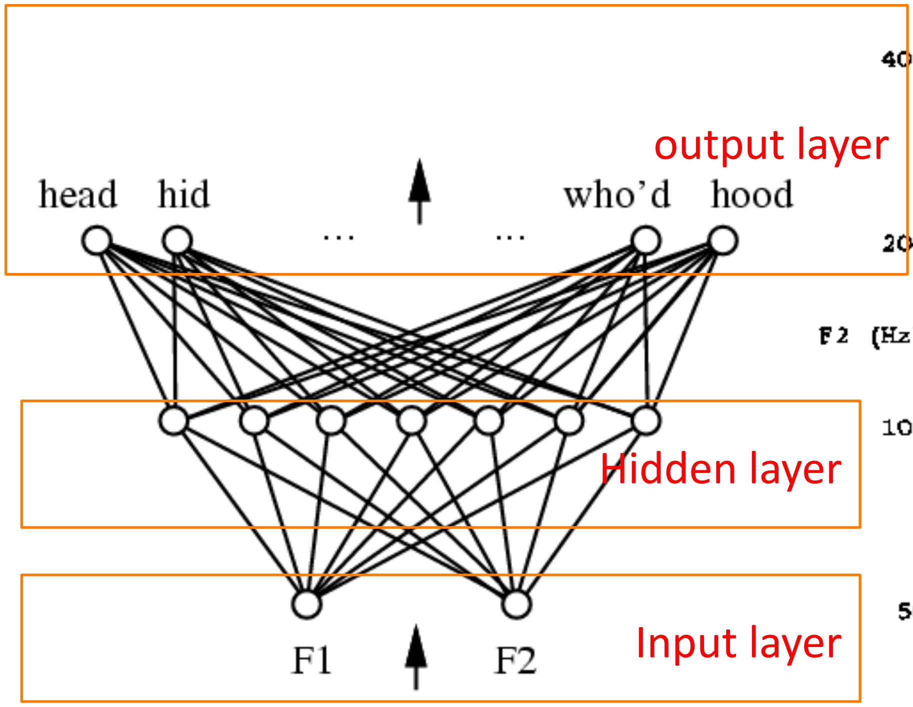
- f might be **non-linear** function
- X (vector of) continuous and/or discrete vars
- Y (vector of) continuous and/or discrete vars

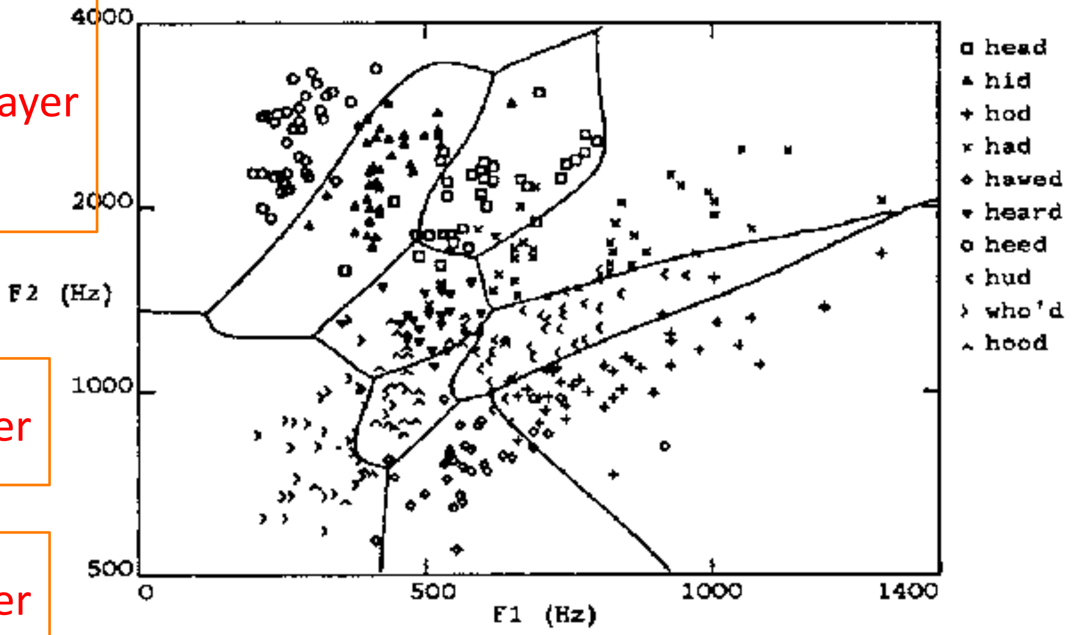
- Represent f by *network* of logistic units
- Each unit is a logistic function

$$unit\ output = \frac{1}{1 + exp(w_0 + \sum_i w_i x_i)}$$

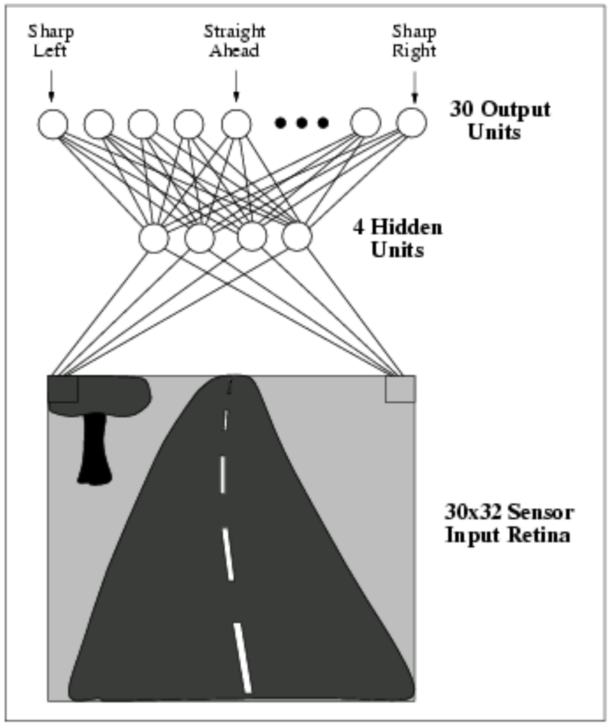
- MLE: train weights of all units to minimize sum of squared errors of predicted network outputs
- MAP: train to minimize sum of squared errors plus weight magnitudes

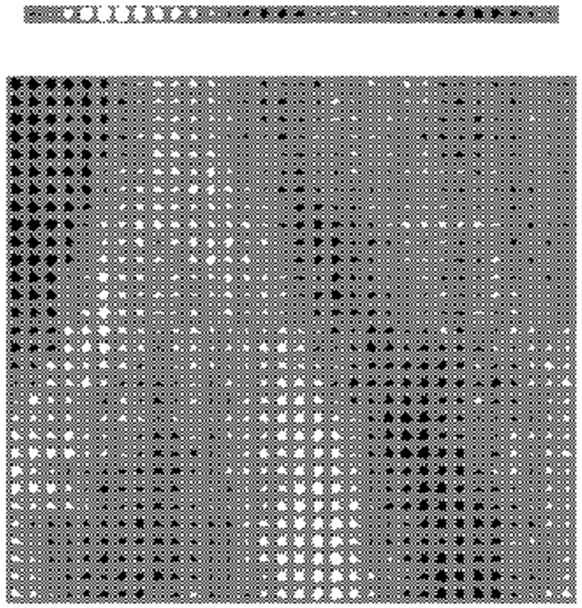
Multilayer Networks of Sigmoid Units





ALVINN
[Pomerleau 1993]





Connectionist Models

Consider humans:

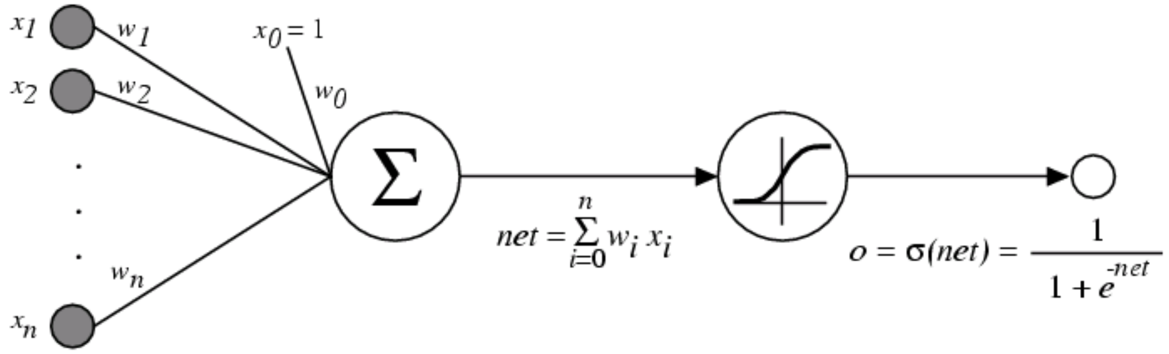
- Neuron switching time ˜ .001 second
- Number of neurons ˜ 10^{10}
- Connections per neuron ˜ 10^{4-5}
- Scene recognition time ˜ .1 second
- 100 inference steps doesn't seem like enough

$\rightarrow$ much parallel computation

Properties of artificial neural nets (ANN's):

- Many neuron-like threshold switching units
- Many weighted interconnections among units
- Highly parallel, distributed process

Sigmoid Unit


x_1
x_2
x_n
w_1
w_2
w_n
$x_0 = 1$
w_0
Σ
$net = \sum_{i=0}^{n} w_i x_i$
$o = \sigma(net) = \frac{1}{1 + e^{-net}}$

$\sigma(x)$ is the sigmoid function

$$\frac{1}{1+e^{-x}}$$

Nice property: $\frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))$

We can derive gradient decent rules to train

- One sigmoid unit
- *Multilayer networks* of sigmoid units $\rightarrow$ Backpropagation

M(C)LE Training for Neural Networks

- Consider regression problem f:X→Y , for scalar Y
  - $y = f(x) + \varepsilon$

assume noise $N(0,\sigma_\varepsilon)$, iid

deterministic

- Let's maximize the conditional data likelihood

$$W \leftarrow \arg\max_W \ \ln \prod_l P(Y^l | X^l, W)$$

$$W \leftarrow \arg\min_W \ \sum_l (y^l - \widehat{f}(x^l))^2$$

Learned neural network

MAP Training for Neural Networks

- Consider regression problem f:X→Y , for scalar Y
  - $y = f(x) + \varepsilon$

assume noise $N(0,\sigma_\varepsilon)$, iid

deterministic

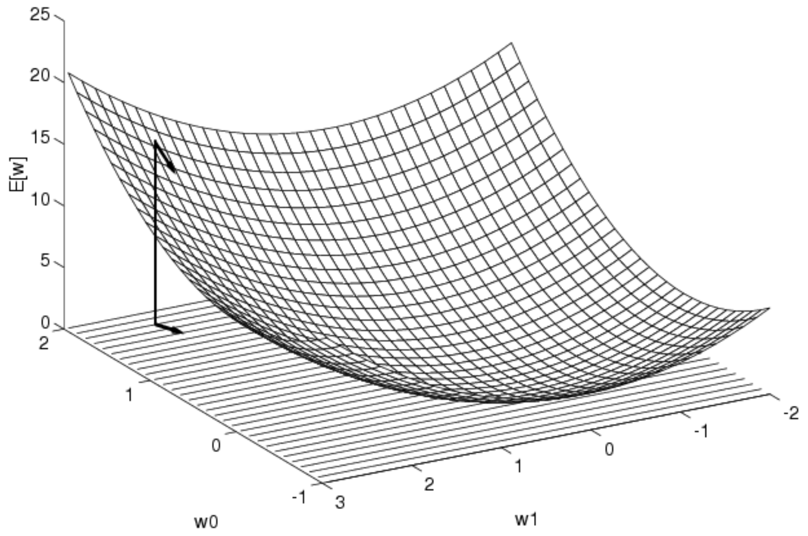
- Let's maximize the posterior (MAP)

$$W \leftarrow \arg\max_W \ \ln \ P(W) \prod_l P(Y^l | X^l, W)$$

$$W \leftarrow \arg\min_W \ \left[c \sum_i w_i^2 \right] + \left[\sum_l (y^l - \hat{f}(x^l))^2 \right]$$

$\ln P(W) \Leftrightarrow c \sum_i w_i^2$

Gradient Descent



Gradient

$$\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots \frac{\partial E}{\partial w_n}\right]$$

Training rule:

$$\Delta \vec{w} = -\eta \nabla E[\vec{w}]$$

i.e.,

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

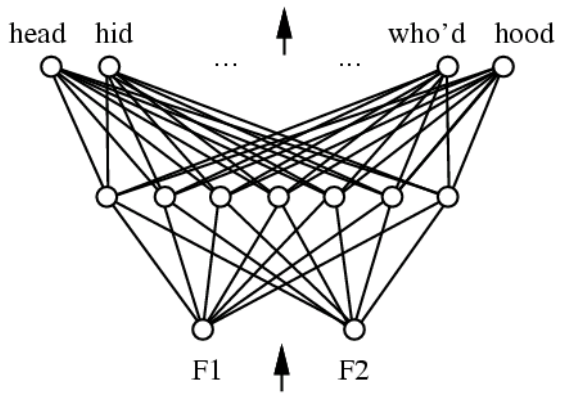
Incremental (Stochastic) Gradient Descent

Batch mode Gradient Descent:
Do until satisfied

1. Compute the gradient $\nabla E_D[\vec{w}]$
2. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_D[\vec{w}]$

Incremental mode Gradient Descent:
Do until satisfied

- For each training example d in D
  1. Compute the gradient $\nabla E_d[\vec{w}]$
  2. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_d[\vec{w}]$



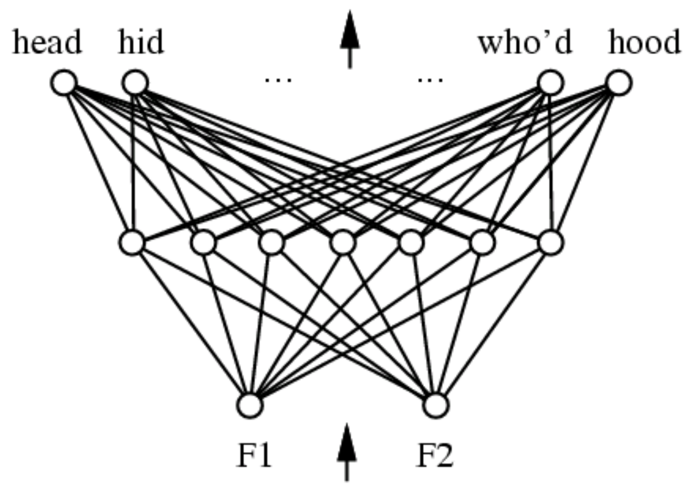
$$E_D[\vec{w}] \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

$$E_d[\vec{w}] \equiv \frac{1}{2} (t_d - o_d)^2$$

t_d = target output
o_d = observed unit output

Incremental Gradient Descent can approximate *Batch Gradient Descent* arbitrarily closely if η made small enough

Backpropagation Algorithm (MLE)



Forward propagation

Initialize all weights to small random numbers.
Until satisfied, Do

- For each training example, Do

1. Input the training example to the network and compute the network outputs
2. For each output unit k

$$\delta_k \leftarrow o_k(1-o_k)(t_k-o_k)$$

3. For each hidden unit h

$$\delta_h \leftarrow o_h(1-o_h)\sum_{k \in outputs} w_{h,k}\delta_k$$

4. Update each network weight $w_{i,j}$

$$w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j}$$

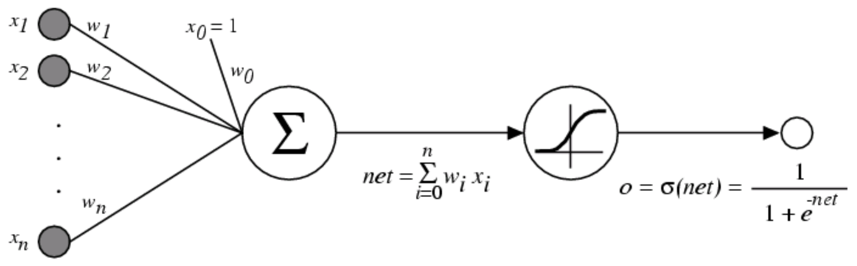
where

$$\Delta w_{i,j} = \eta \delta_j x_i$$

x_d = input
t_d = target output
o_d = observed unit output
w_{ij} = wt from i to j

Backward propagation

Error Gradient for a Sigmoid Unit


x_1
x_2
x_n
w_1
w_2
w_n
$x_0 = 1$
w_0
Σ
$net = \sum_{i=0}^{n} w_i x_i$
$o = \sigma(net) = \frac{1}{1+e^{-net}}$

$$
\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 \\
&= \frac{1}{2} \sum_d \frac{\partial}{\partial w_i} (t_d - o_d)^2 \\
&= \frac{1}{2} \sum_d 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d) \\
&= \sum_d (t_d - o_d) \left(- \frac{\partial o_d}{\partial w_i} \right) \\
&= - \sum_d (t_d - o_d) \frac{\partial o_d}{\partial net_d} \frac{\partial net_d}{\partial w_i}
\end{aligned}
$$

x_d = input
t_d = target output
o_d = observed unit output
w_i = weight i

But we know:

$$\frac{\partial o_d}{\partial net_d} = \frac{\partial \sigma(net_d)}{\partial net_d} = o_d(1 - o_d)$$

$$\frac{\partial net_d}{\partial w_i} = \frac{\partial (\vec{w} \cdot \vec{x}_d)}{\partial w_i} = x_{i,d}$$

So:

$$\frac{\partial E}{\partial w_i} = - \sum_{d \in D} (t_d - o_d) o_d (1 - o_d) x_{i,d}$$

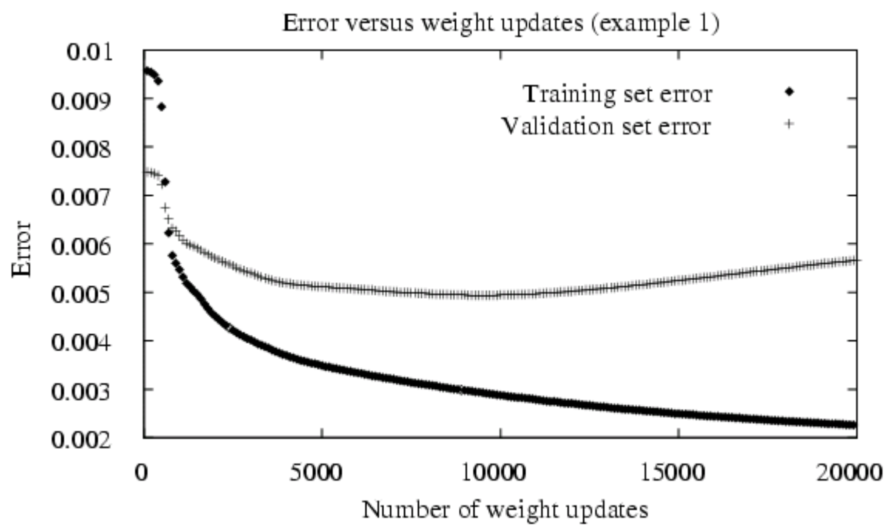
More on Backpropagation

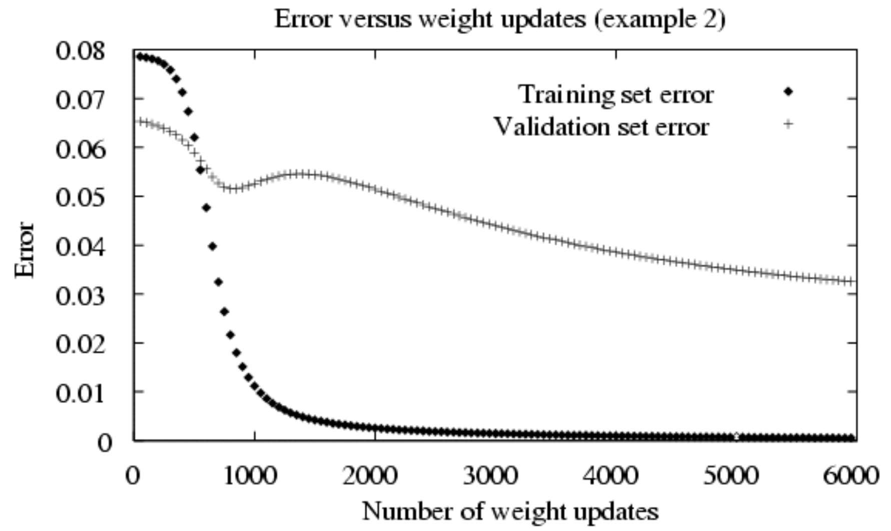
- Gradient descent over entire *network* weight vector
- Easily generalized to arbitrary directed graphs
- Will find a local, not necessarily global error minimum
  - In practice, often works well (can run multiple times)
- Often include weight *momentum* α

$$\Delta w_{i,j}(n) = \eta \delta_j x_{i,j} + \alpha \Delta w_{i,j}(n-1)$$

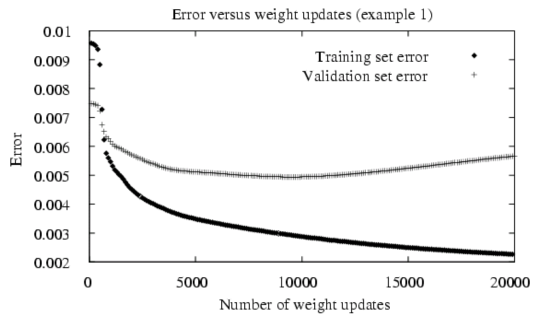
- Minimizes error over *training* examples
  - Will it generalize well to subsequent examples?
- Training can take thousands of iterations $\rightarrow$ slow!
- Using network after training is very fast

Overfitting in ANNs





Dealing with Overfitting



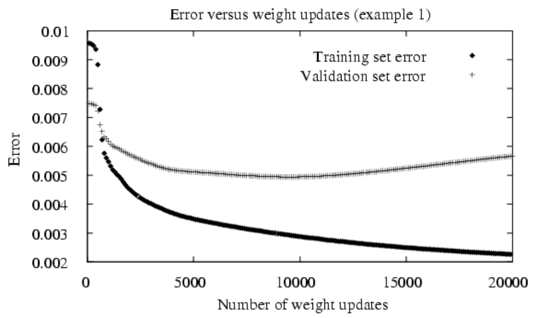
Our learning algorithm involves a parameter

n=number of gradient descent iterations

How do we choose n to optimize future error?

e.g. the n that minimizes error rate of neural net over future data

Dealing with Overfitting



Our learning algorithm involves a parameter

n=number of gradient descent iterations

How do we choose n to optimize future error?

- Separate available data into training and validation set
- Use training to perform gradient descent
- n $\leftarrow$ number of iterations that optimizes validation set error

$\rightarrow$ *gives unbiased estimate of optimal n*

(but a biased estimate of true error)

K-Fold Cross Validation

Idea: train multiple times, leaving out a disjoint subset of data each time for test.
Average the test set accuracies.

____

```
Partition data into K disjoint subsets
For k=1 to K
    testData = kth subset
    h ← classifier trained* on all data except for testData
    accuracy(k) = accuracy of h on testData
end
FinalAccuracy = mean of the K recorded testset accuracies
```

* might withhold some of this to choose number of gradient decent steps

Leave-One-Out Cross Validation

This is just k-fold cross validation leaving out one example each iteration

____

```
Partition data into K disjoint subsets, each containing one example
For k=1 to K
    testData = kth subset
    h ← classifier trained* on all data except for testData
    accuracy(k) = accuracy of h on testData
end
FinalAccuracy = mean of the K recorded testset accuracies
```

* might withhold some of this to choose number of gradient decent steps

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 arbitrarily 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].

Convergence of Backpropagation

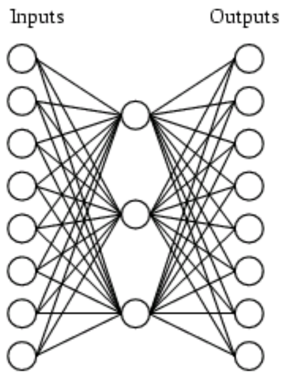
Gradient descent to some local minimum

- Perhaps not global minimum...
- Add momentum
- Stochastic gradient descent
- Train multiple nets with different inital weights

Nature of convergence

- Initialize weights near zero
- Therefore, initial networks near-linear
- Increasingly non-linear functions possible as training progresses

Learning Hidden Layer Representations



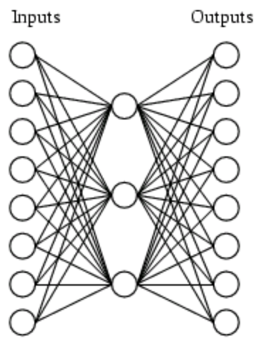
A target function:

Input		Output
10000000	$\rightarrow$	10000000
01000000	$\rightarrow$	01000000
00100000	$\rightarrow$	00100000
00010000	$\rightarrow$	00010000
00001000	$\rightarrow$	00001000
00000100	$\rightarrow$	00000100
00000010	$\rightarrow$	00000010
00000001	$\rightarrow$	00000001

Can this be learned??

Learning Hidden Layer Representations

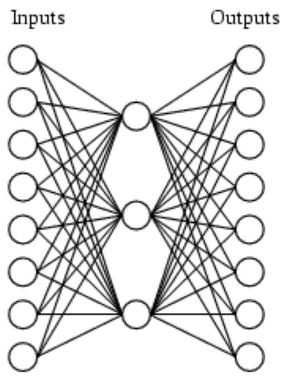
A network:

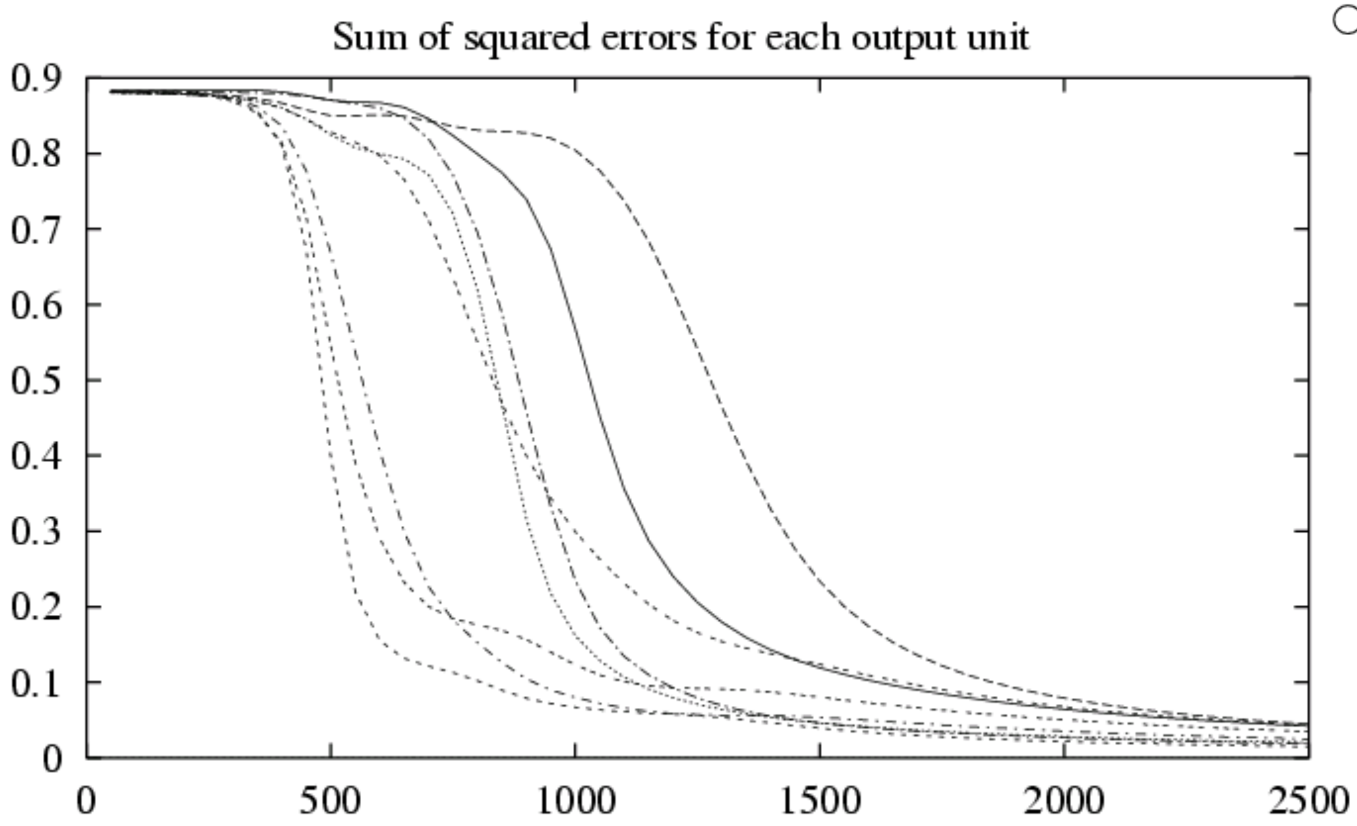


Learned hidden layer representation:

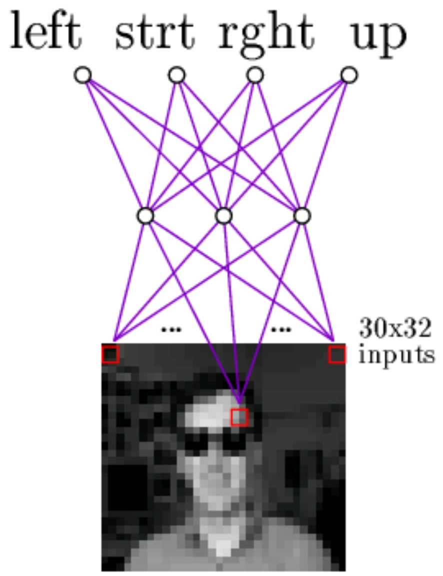
Input		Hidden Values				Output
10000000	→	.89	.04	.08	→	10000000
01000000	→	.01	.11	.88	→	01000000
00100000	→	.01	.97	.27	→	00100000
00010000	→	.99	.97	.71	→	00010000
00001000	→	.03	.05	.02	→	00001000
00000100	→	.22	.99	.99	→	00000100
00000010	→	.80	.01	.98	→	00000010
00000001	→	.60	.94	.01	→	00000001

Training


Inputs
Outputs


Sum of squared errors for each output unit
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
500
1000
1500
2000
2500

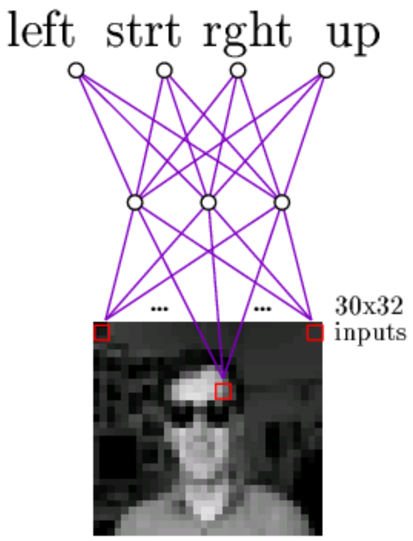
Neural Nets for Face Recognition

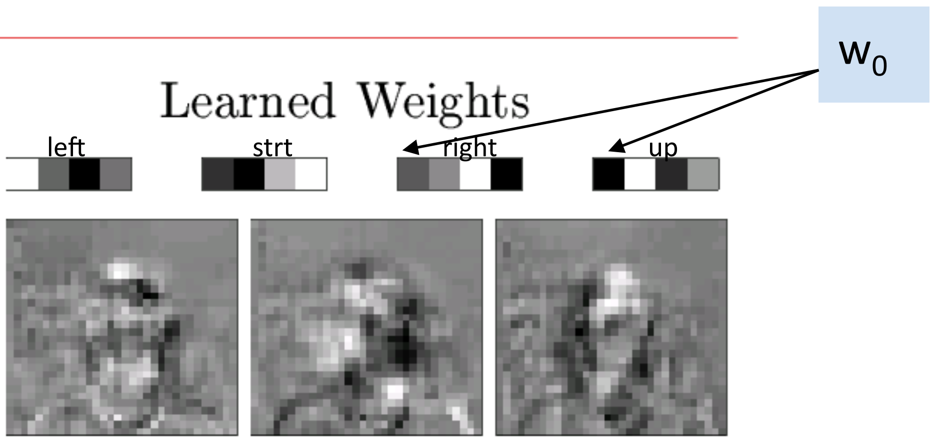


Typical input images

90% accurate learning head pose, and recognizing 1-of-20 faces

Learned Hidden Unit Weights





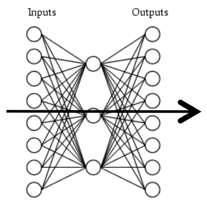
Typical input images

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

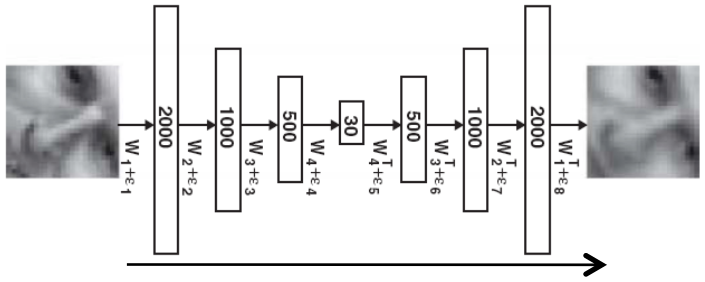
[Hinton & Salakhutdinov, 2006]

Deep Belief Networks

- Problem: training networks with many hidden layers doesn't work very well
  - local minima, very slow training if initialize with zero weights
- Deep belief networks
  - autoencoder networks to learn low dimensional encodings

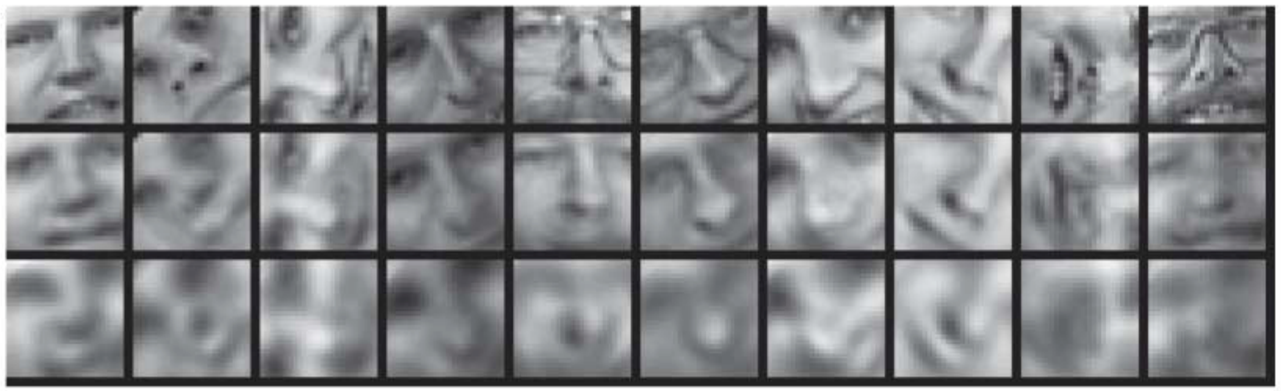


  - but more layers, to learn better encodings



[Hinton & Salakhutdinov, 2006]

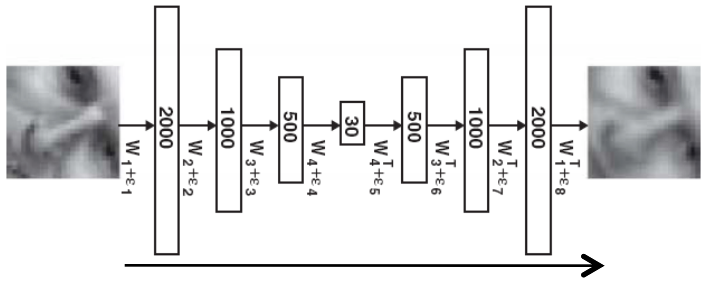
Deep Belief Networks



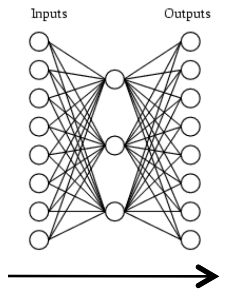
original image

reconstructed from 2000-1000-500-30 DBN

reconstructed from 2000-300, linear PCA



versus

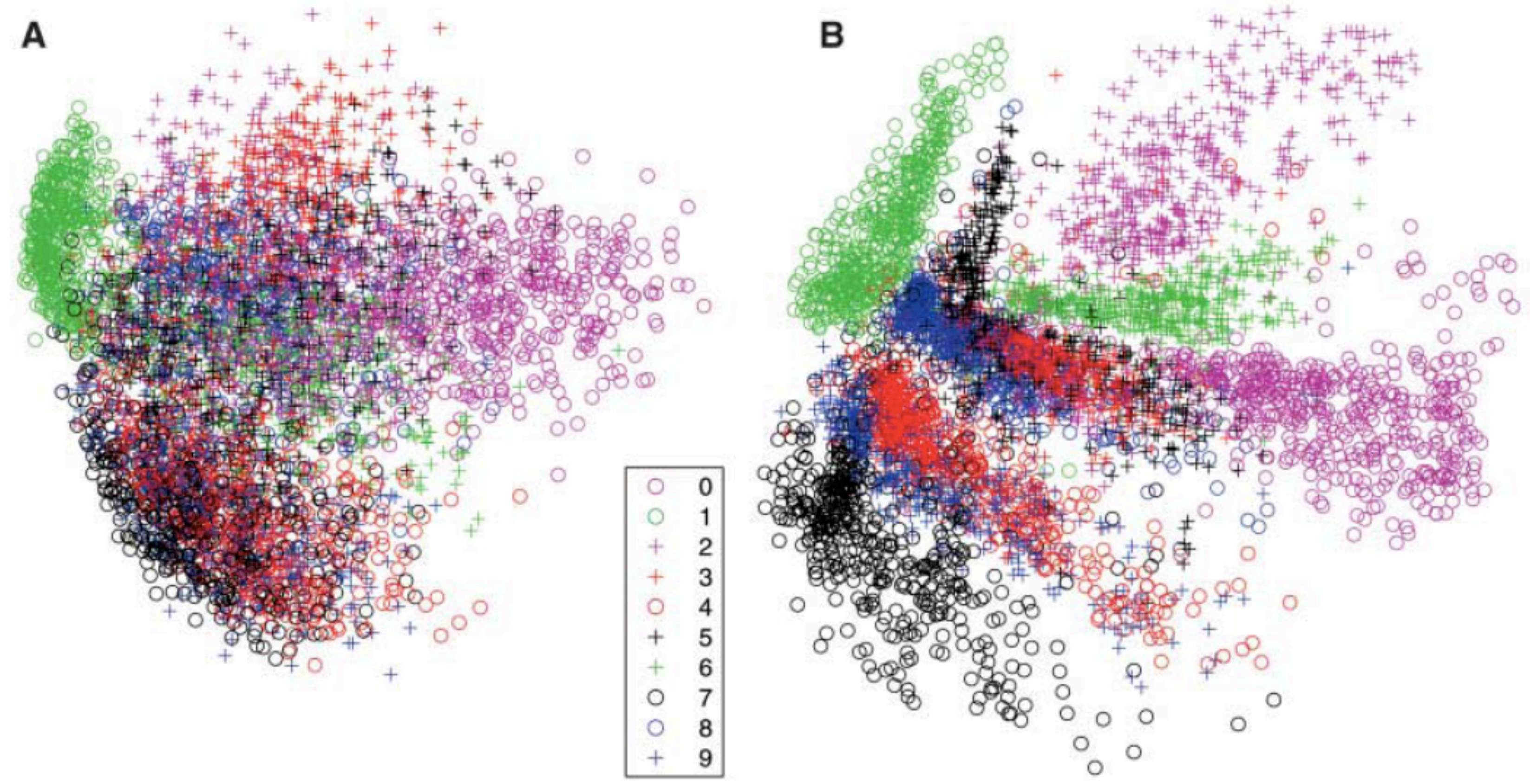


Encoding of digit images in two dimensions

[Hinton & Salakhutdinov, 2006]

784 pixel image -> 2 dimension linear encoding (PCA)

784-1000-500-250-2 DBNet



[Hinton & Salakhutdinov, 2006]

Deep Belief Networks: Training

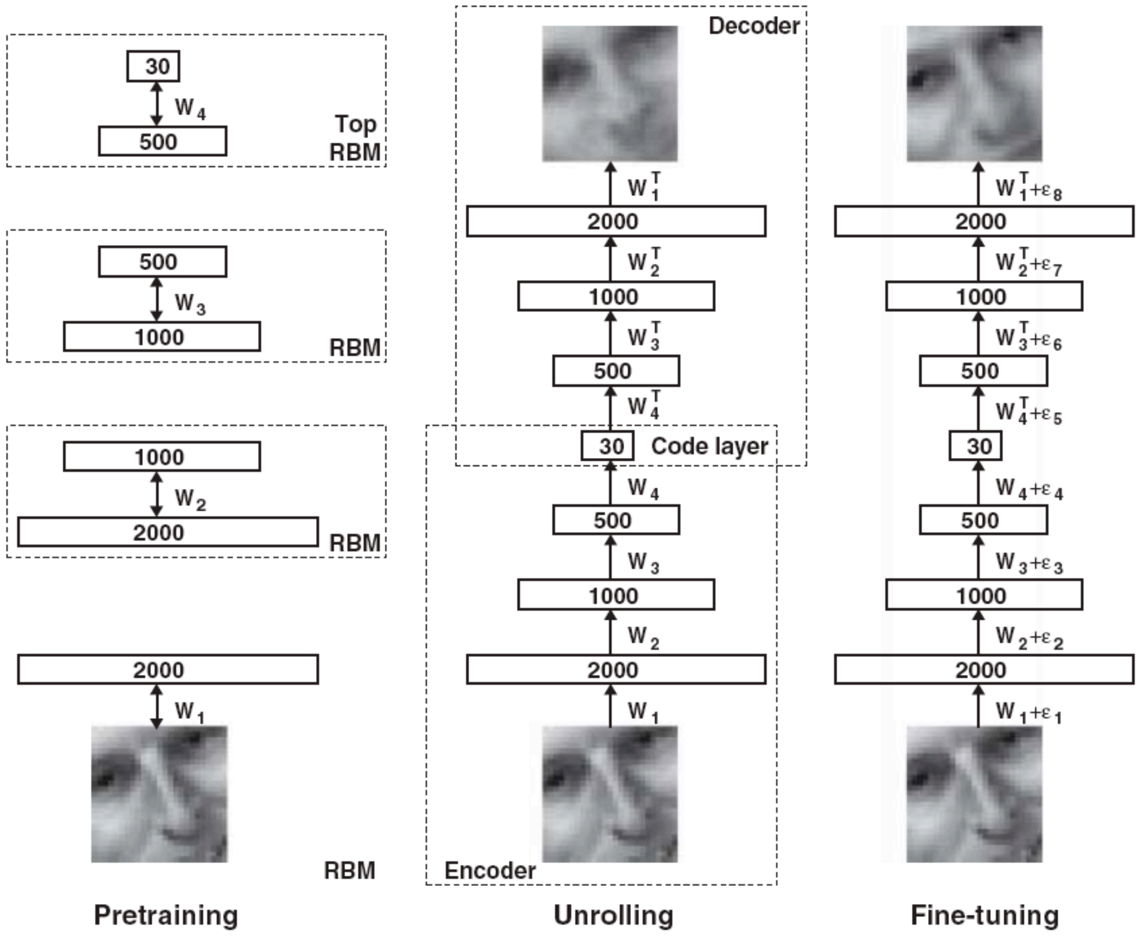
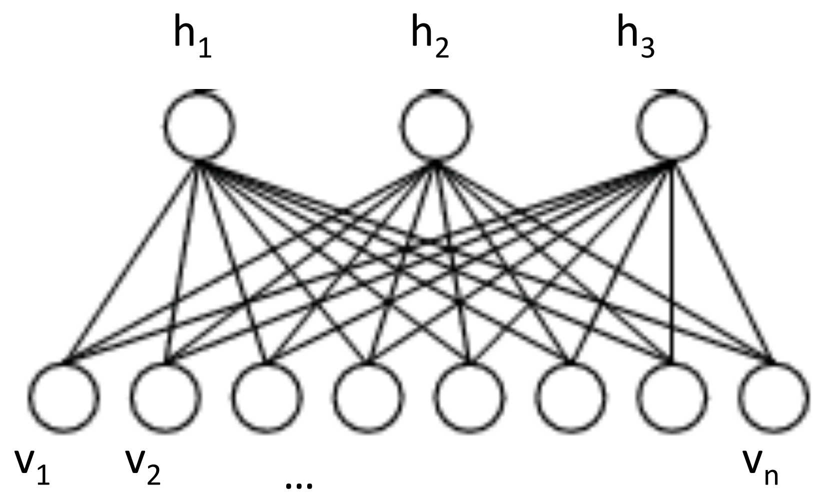


Fig. 1. Pretraining consists of learning a stack of restricted Boltzmann machines (RBMs), each having only one layer of feature detectors. The learned feature activations of one RBM are used as the "data" for training the next RBM in the stack. After the pretraining, the RBMs are "unrolled" to create a deep autoencoder, which is then fine-tuned using backpropagation of error derivatives.

Restricted Boltzman Machine

- Bipartite graph, logistic activation
- Inference: fill in any nodes, estimate other nodes
- consider v_i, h_j are boolean variables



$$P(h_j = 1|\mathbf{v}) = \frac{1}{1 + \exp(\sum_i w_{ij} v_i)}$$

$$P(v_i = 1|\mathbf{h}) = \frac{1}{1 + \exp(\sum_j w_{ij} h_j)}$$

Very Large Scale Use of DBN's [Quoc Le, et al., *ICML*, 2012]

Data: 10 million 200x200 unlabeled images, sampled from YouTube

Training: use 1000 machines (16000 cores) for 1 week

Learned network: 3 multi-stage layers, 1.15 billion parameters

Achieves 15.8% accuracy classifying 1 of 20k categories in ImageNet data

Images that most excite the feature:

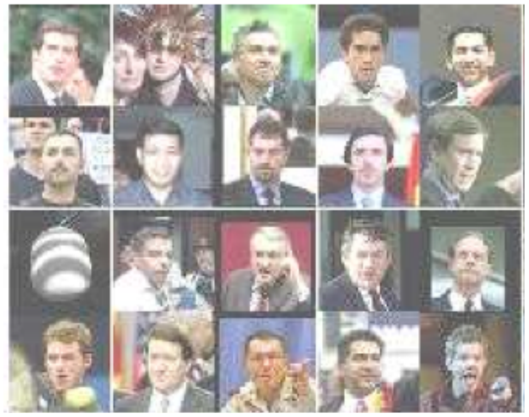
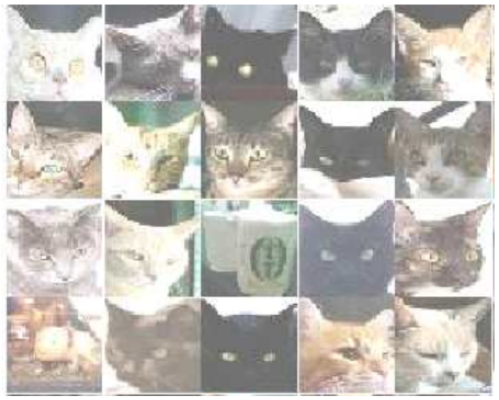
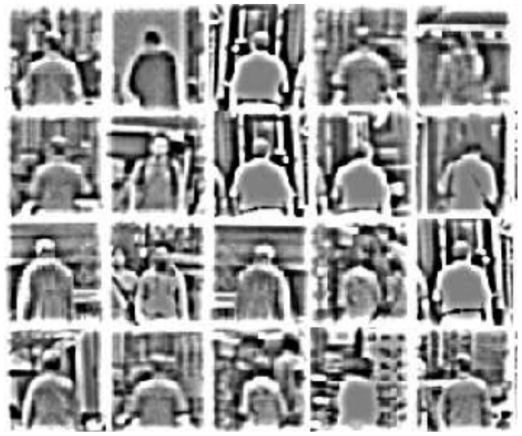
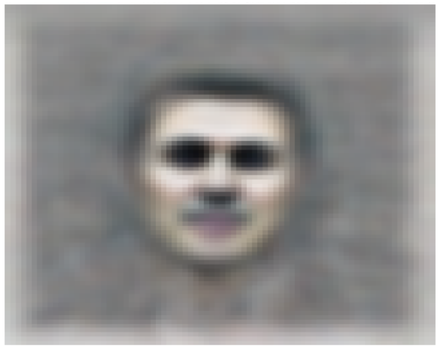
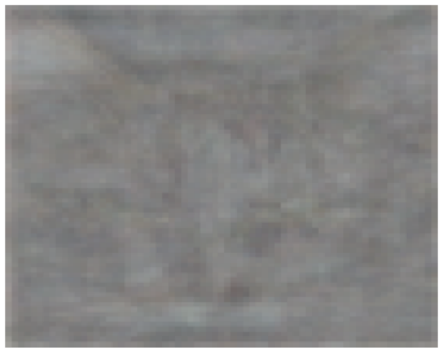
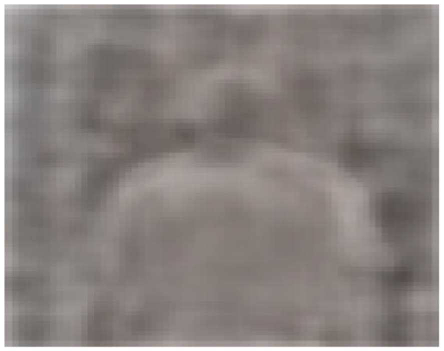
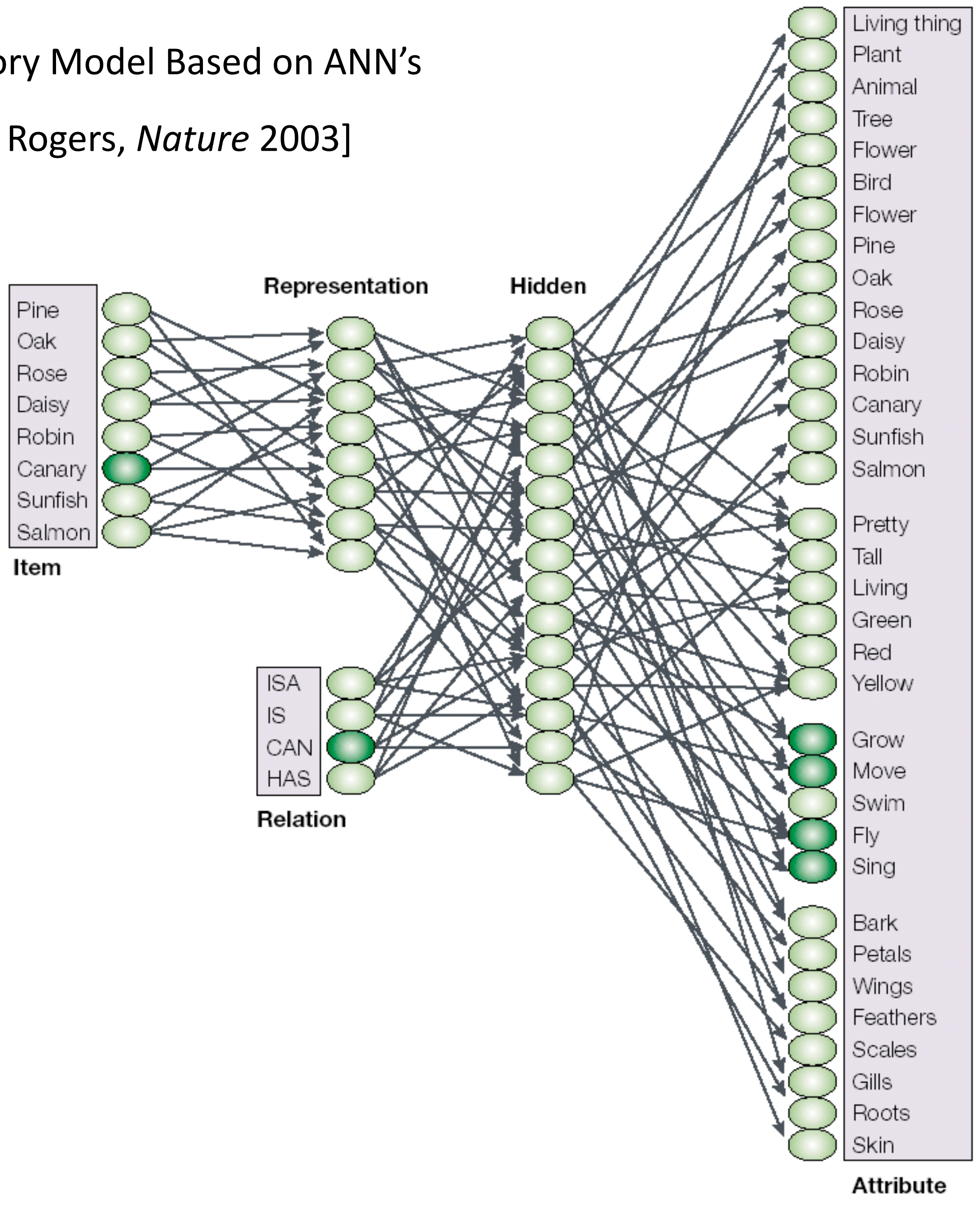




Image synthesized to most excite the feature:

Semantic Memory Model Based on ANN's

[McClelland & Rogers, *Nature* 2003]



No hierarchy given.

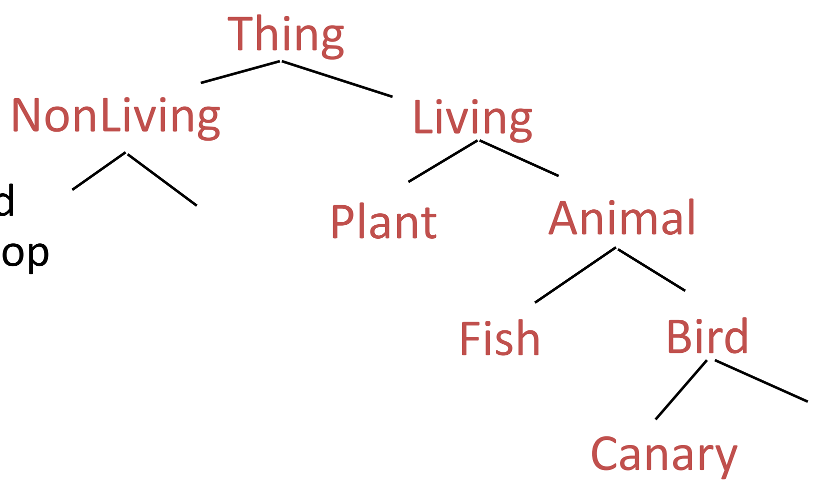
Train with assertions, e.g., Can(Canary,Fly)

Humans act as though they have a hierarchical memory organization

1. Victims of Semantic Dementia progressively lose knowledge of objects

   But they lose specific details first, general properties later, suggesting hierarchical memory organization

2. Children appear to learn general categories and properties first, following the same hierarchy, top down*.



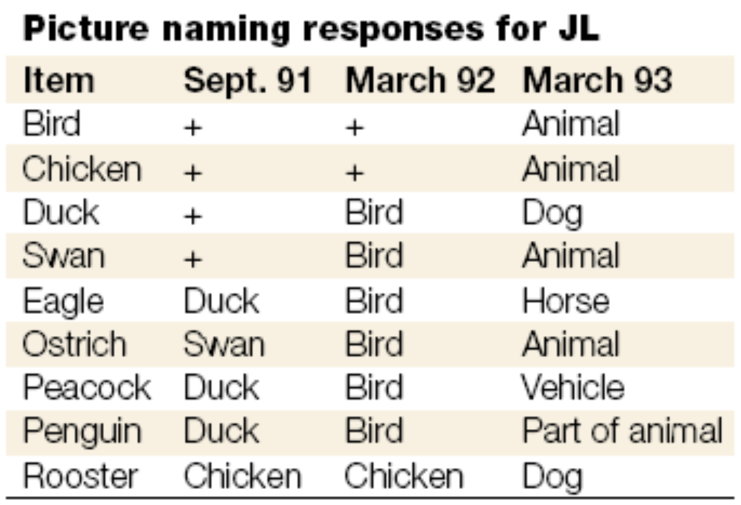
Question: What learning mechanism could produce this emergent hierarchy?

* some debate remains on this.

Memory deterioration follows semantic hierarchy

[McClelland & Rogers, *Nature* 2003]

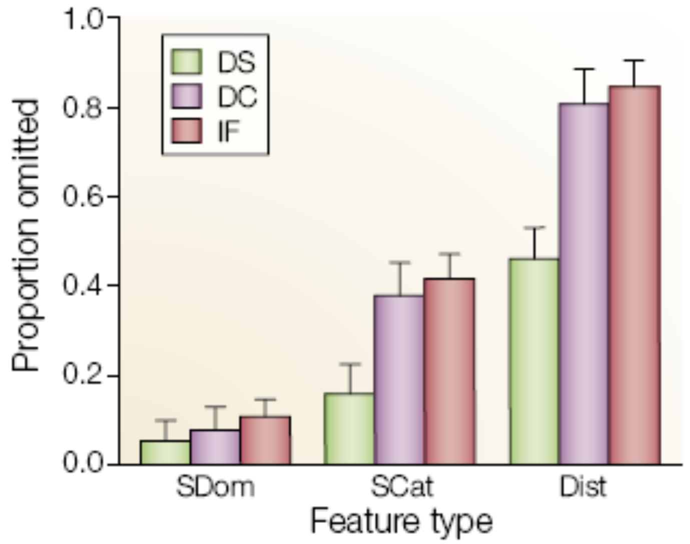
a



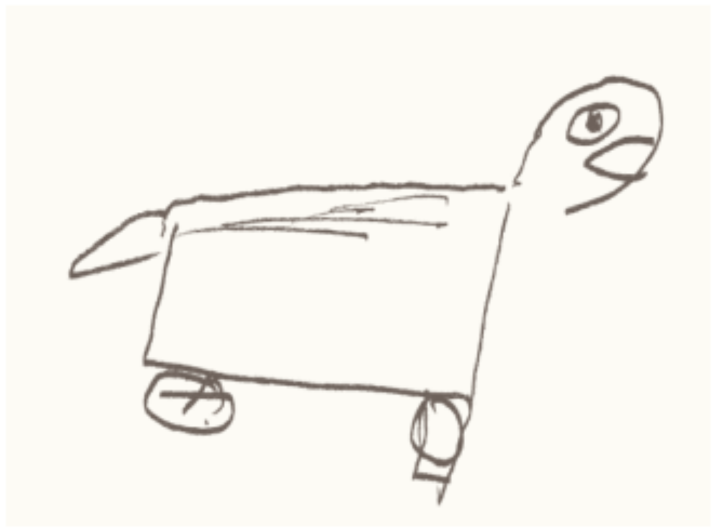
Picture naming responses for JL

Item	Sept. 91	March 92	March 93
Bird	+	+	Animal
Chicken	+	+	Animal
Duck	+	Bird	Dog
Swan	+	Bird	Animal
Eagle	Duck	Bird	Horse
Ostrich	Swan	Bird	Animal
Peacock	Duck	Bird	Vehicle
Penguin	Duck	Bird	Part of animal
Rooster	Chicken	Chicken	Dog

b



c IF's delayed copy of a camel



d DC's delayed copy of a swan

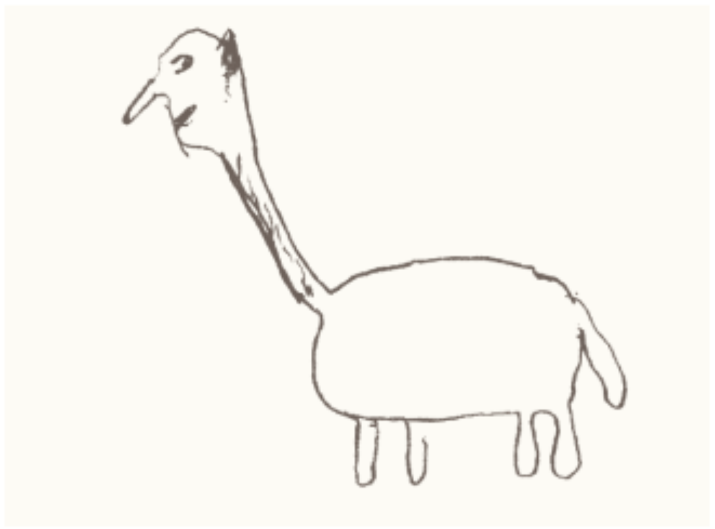


Figure 2 | **Evidence of conceptual disintegration in semantic dementia. a** | Naming responses given by patient JL to pictures of birds (drawn from a set of line drawings for which

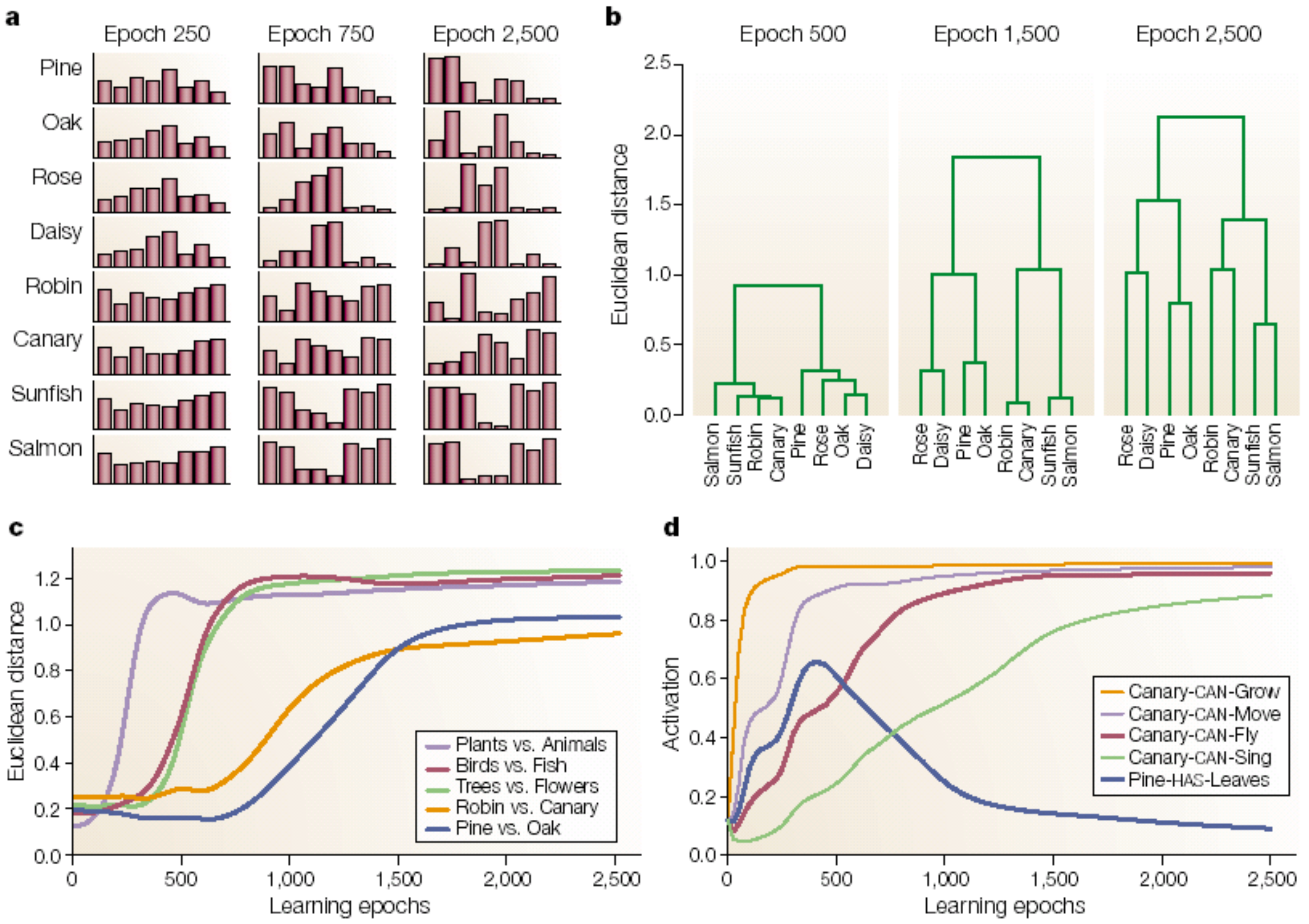
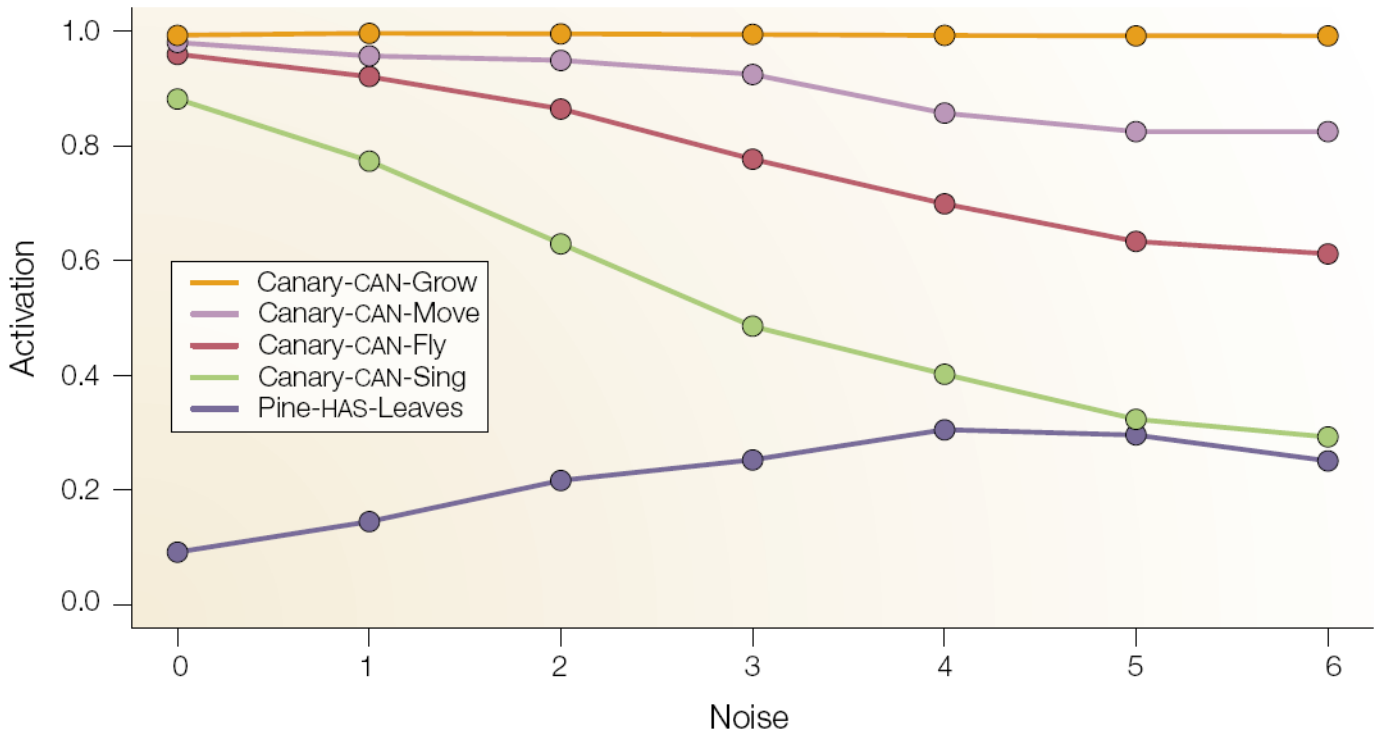


Figure 4 | **The process of differentiation of conceptual representations.** The representations are those seen in the feedforward network model shown in FIG. 3. **a** | Acquired patterns of activation that represent the eight objects in the training set at three points in the learning process (epochs 250, 750 and 2,500). Early in learning, the patterns are undifferentiated; the first difference to appear is between plants and animals. Later, the patterns show clear differentiation at both the superordinate (plant–animal) and intermediate (bird–fish/tree–flower) levels. Finally, the individual concepts are differentiated, but the overall hierarchical organization of the similarity structure remains. **b** | A standard hierarchical clustering analysis program has been used to visualize the similarity structure in the patterns shown in **a**. The algorithm searches the patterns to find the two that are the closest, according to a Euclidean distance measure, creates a node in the tree at a vertical position corresponding to the distance between them, replaces the two patterns with their average, and then iterates until one grand average pattern remains. **c** | Pairwise distances between representations of

ANN Also Models Progressive Deterioration

[McClelland & Rogers, *Nature* 2003]



average effect of noise in inputs to hidden layers

What you should know: Artificial Neural Networks

- Highly non-linear regression/classification
- Vector-valued inputs and outputs
- Potentially millions of parameters to estimate
- Hidden layers learn intermediate representations
- Actively used to model distributed computation in brain

- Backpropagation algorithm for learning
- Gradient descent, local minima problems
- Overfitting and how to deal with it

- Many extensions