Neural Networks

Required reading:

- Neural nets: Mitchell chapter 4

Optional reading:

- Bias/Variance error decomposition: Bishop: 9.1, 9.2

Machine Learning 10-701

Tom M. Mitchell
Center for Automated Learning and Discovery
Carnegie Mellon University

October 4, 2005

Today:

- Finish up
  - MLE vs MAP for logisitic regression
  - Generative/Discriminative classifiers

- Artificial neural networks

MLE vs MAP

- Maximum conditional likelihood estimate

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

$$w_i \leftarrow w_i + \eta \sum_l X_i^l (Y^l - \hat{P}(Y^l = 1|X^l, W))$$

- Maximum a po

Gaussian P(W) = N(0,σI)

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

$$w_i \leftarrow w_i - \eta\lambda w_i + \eta \sum_l X_i^l (Y^l - \hat{P}(Y^l = 1|X^l, W))$$

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

MLE vs MAP

- Maximum conditional likelihood estimate

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

$$w_i \leftarrow w_i + \eta \sum_l X_i^l (Y^l - \hat{P}(Y^l = 1|X^l, W))$$

- Maximum a po

Gaussian P(W) = N(<μ_1 ... μ_n > ,σI)

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

$$w_i \leftarrow w_i - \eta\lambda(w_i - \mu_i) + \eta \sum_l X_i^l (Y^l - \hat{P}(Y^l = 1|X^l, W))$$

ln P(W) $\leftrightarrow$ c $\sum_i$ (w_i - μ_i)^2

Generative vs. Discriminative Classifiers

Training classifiers involves estimating f: X → Y, or P(Y|X)

Generative classifiers:

- Assume some functional form for P(X|Y), P(X)
- Estimate parameters of P(X|Y), P(X) from training data
- Use Bayes rule to calculate P(Y|X= x_i)

Discriminative classifiers:

- Assume some functional form for P(Y|X)
- Estimate parameters of P(Y|X) from training data

Naïve Bayes vs Logistic Regression

Consider Y boolean, X_i continuous, $X=<X_1 \ldots X_n>$

Number of parameters:

- NB: 4n +1
- LR: n+1

Estimation method:

- NB parameter estimates are uncoupled
- LR parameter estimates are coupled

Naïve Bayes vs. Logistic Regression

[Ng & Jordan, 2002]

- Generative and Discriminative classifiers

- Asymptotic comparison (# training examples → infinity)
  - when model correct
    - GNB, LR produce identical classifiers
  - when model incorrect
    - LR is less biased – does not assume cond indep.
    - therefore expected to outperform GNB

Naïve Bayes vs. Logistic Regression

- Generative and Discriminative classifiers

- Non-asymptotic analysis (see [Ng & Jordan, 2002])
  - convergence rate of parameter estimates (slightly oversimplified – see paper for bounds)
    - GNB order log n (where n = # of attributes in X)
    - LR order n

  GNB converges more quickly to its (perhaps less helpful) asymptotic estimates

Some experiments from UCI data sets

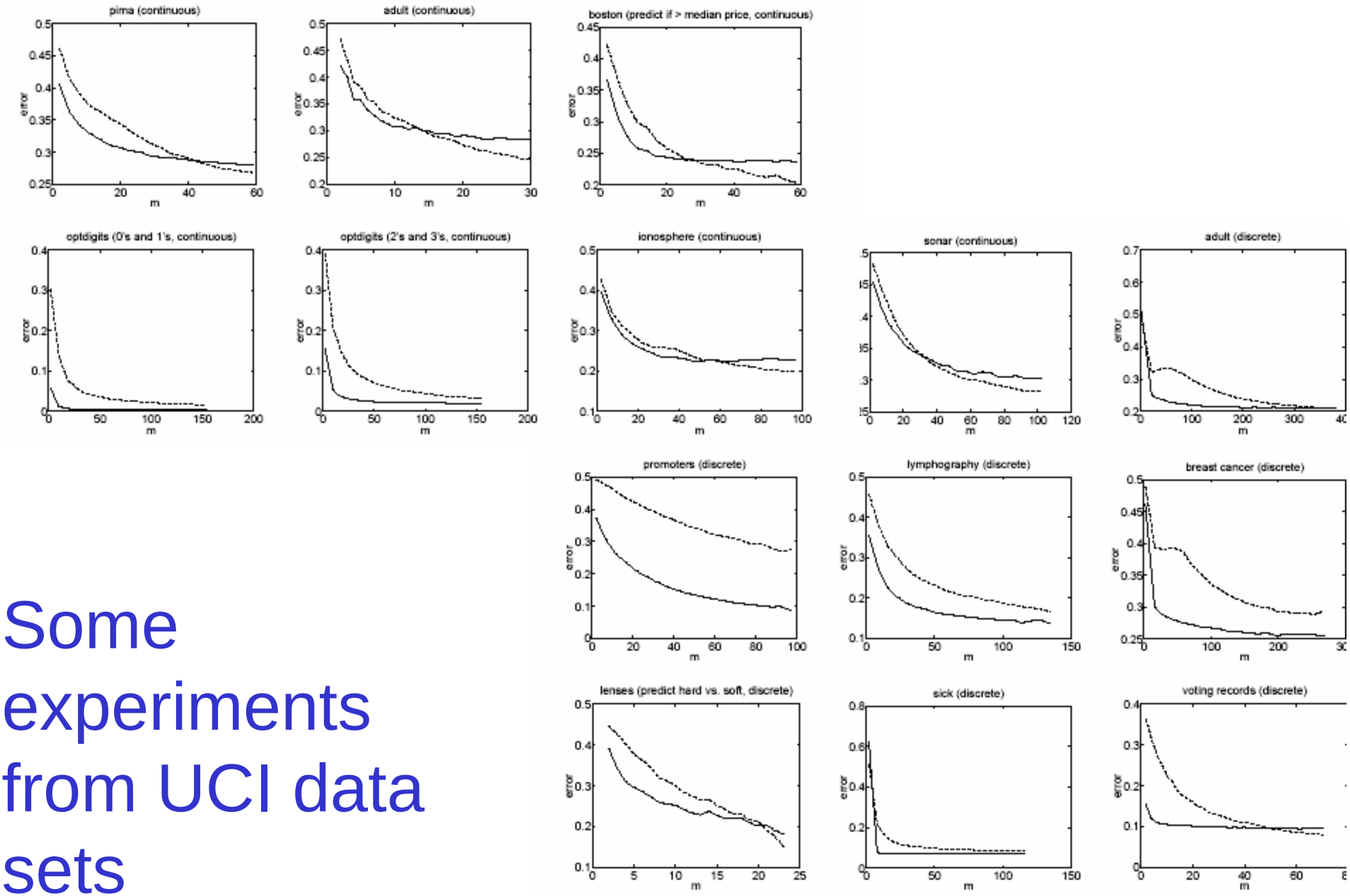


Figure 1: Results of 15 experiments on datasets from the UCI Machine Learnin repository. Plots are of generalization error vs. m (averaged over 1000 randor train/test splits). Dashed line is logistic regression; solid line is naive Bayes.

What you should know:

- Logistic regression
  - Functional form follows from Naïve Bayes assumptions
  - But training procedure picks parameters without the conditional independence assumption
  - MLE training: pick W to maximize P(Y | X, W)
  - MAP training: pick W to maximize P(W | X,Y)
    - 'regularization'
- Gradient ascent/descent
  - General approach when closed-form solutions unavailable
- Generative vs. Discriminative classifiers

Artificial Neural Networks

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 threshold 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 at network outputs

Connectionist Models

Consider humans:

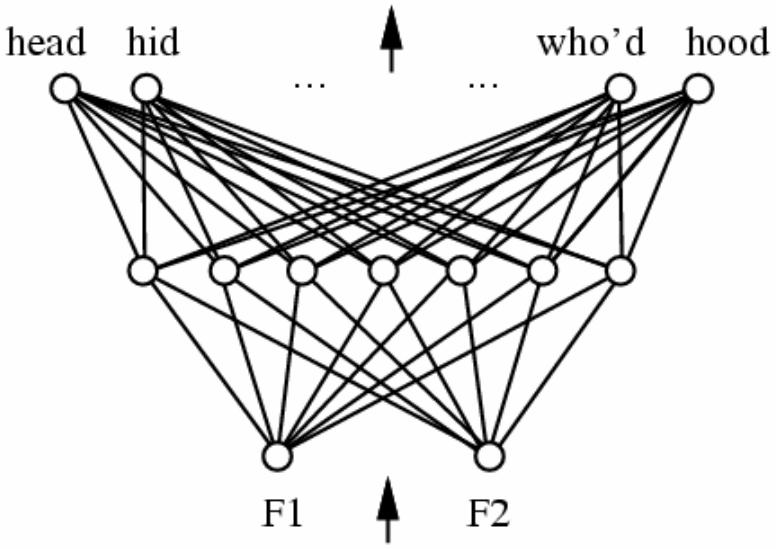
- Neuron switching time ~ .001 second
- Number of neurons ~ 10^{10}
- Connections per neuron ~ 10^{4-5}
- Scene recognition time ~ .1 second
- 100 inference steps doesn't seem like enough

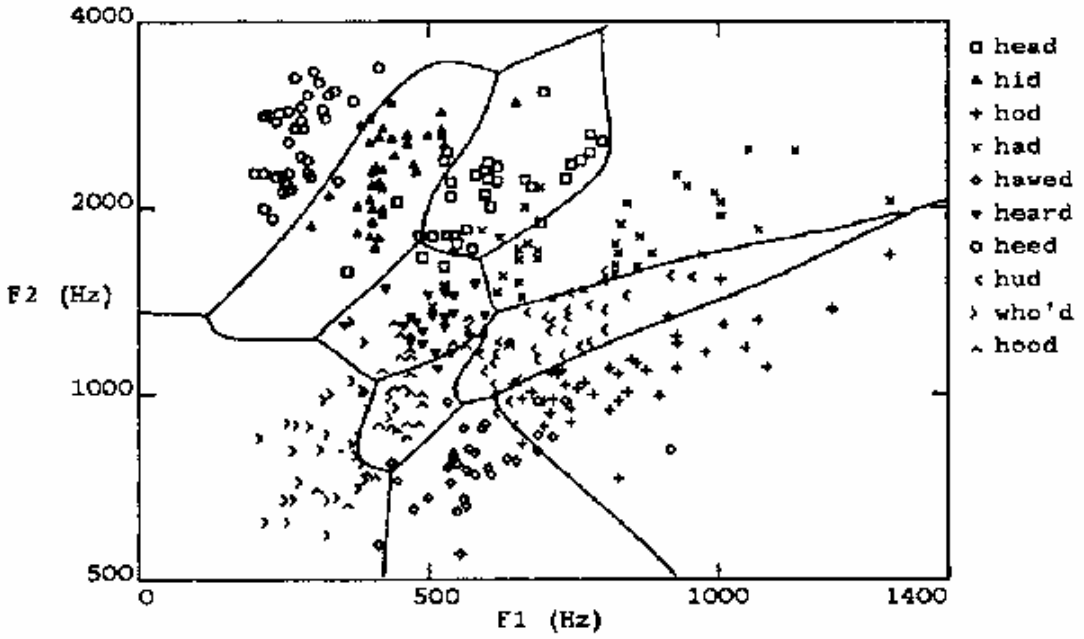
$\rightarrow$ much parallel computation

Properties of artificial neural nets (ANN's):

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

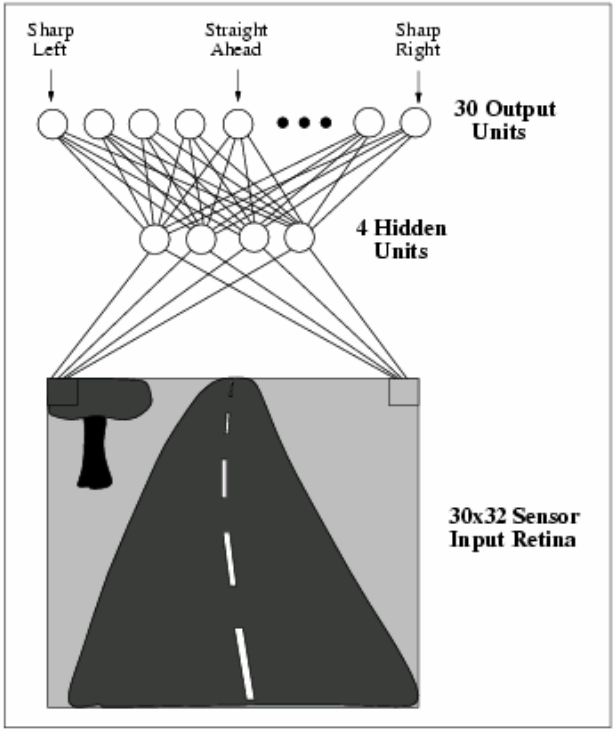
Multilayer Networks of Sigmoid Units

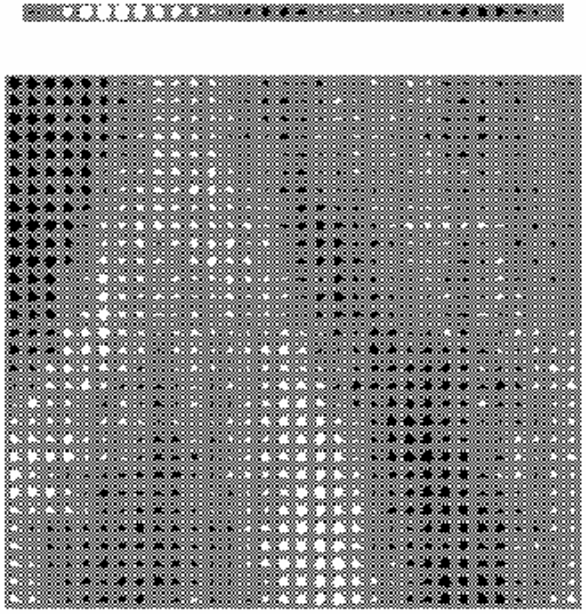




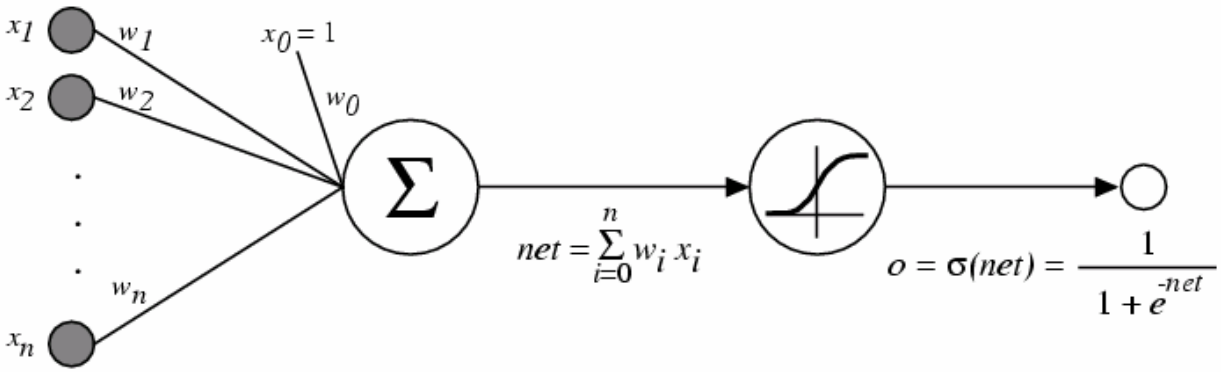
ALVINN

[Pomerleau 1993]





Sigmoid Unit



$\sigma(x)$ 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

MLE Training for Neural Networks

- Consider regression problem f:X→Y , for scalar Y

$y = f(x) + \varepsilon$ ← noise $N(0,\sigma_\varepsilon)$

deterministic

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

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

Learned neural network

MAP Training for Neural Networks

- Consider regression problem f:X→Y , for scalar Y

$y = f(x) + \varepsilon$ ← noise $N(0,\sigma_\varepsilon)$

deterministic

Gaussian P(W) = N(0,σI)

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

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

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

MLE Training for Neural Networks

- Consider regression problem f:X→Y , for Y=<y_1 ... y_N>

$y_i = f_i(x) + \varepsilon_i$

noise N(0,σ) drawn independently for each output y_i

deterministic

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

$$W \leftarrow \arg\min_W \ \sum_l \sum_i (y_i^l - \hat{f}_i(x^l))^2$$

Error Gradient for a Sigmoid Unit

t_d = target output

o_d = observed unit output

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

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:

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

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. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_d[\vec{w}]$

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

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

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

Backpropagation Algorithm

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

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

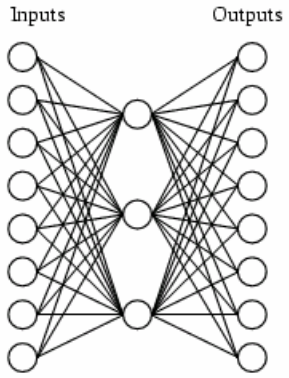
More on Backpropagation

- 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

Learning Hidden Layer Representations



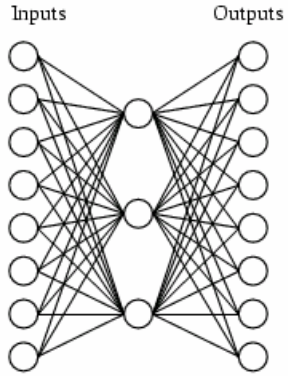
A target function:

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

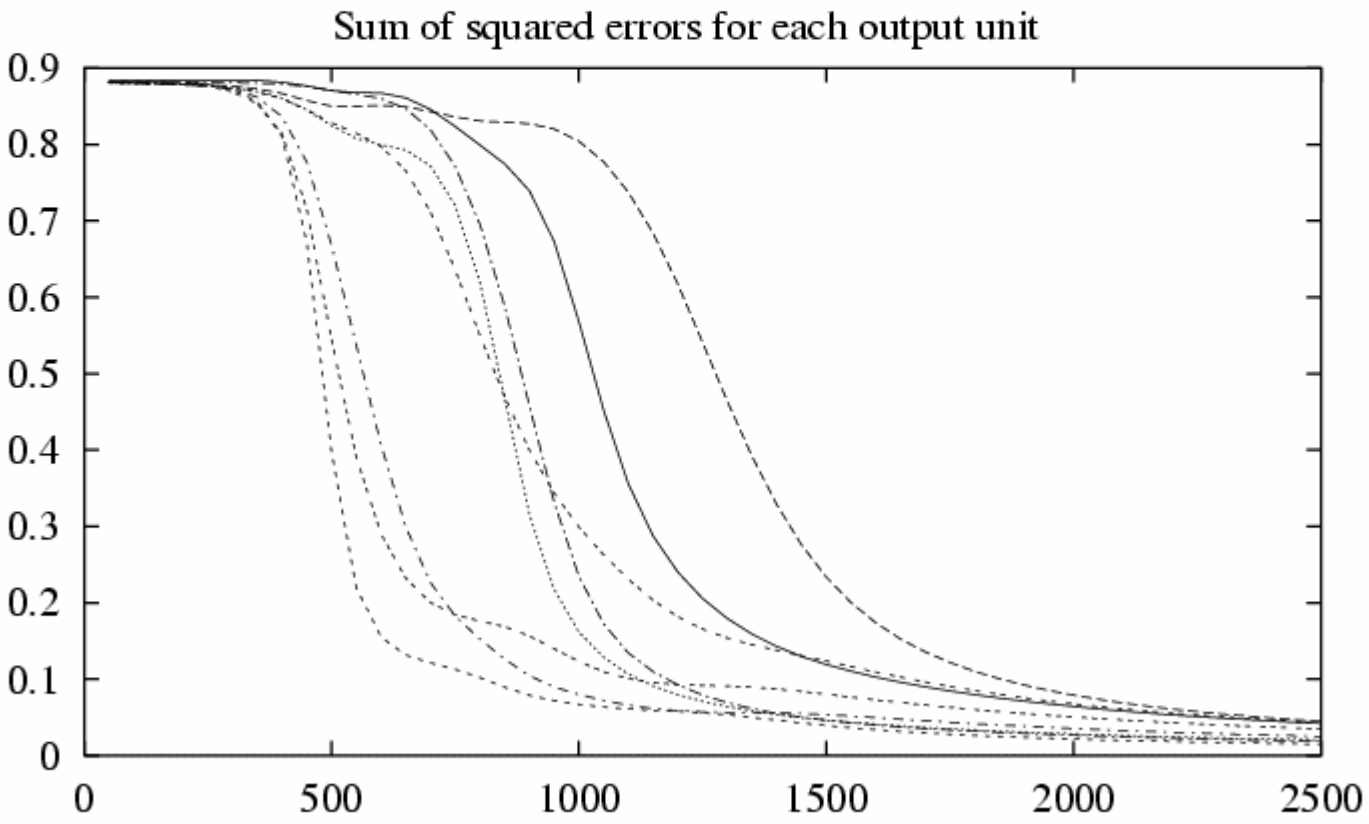
A network:



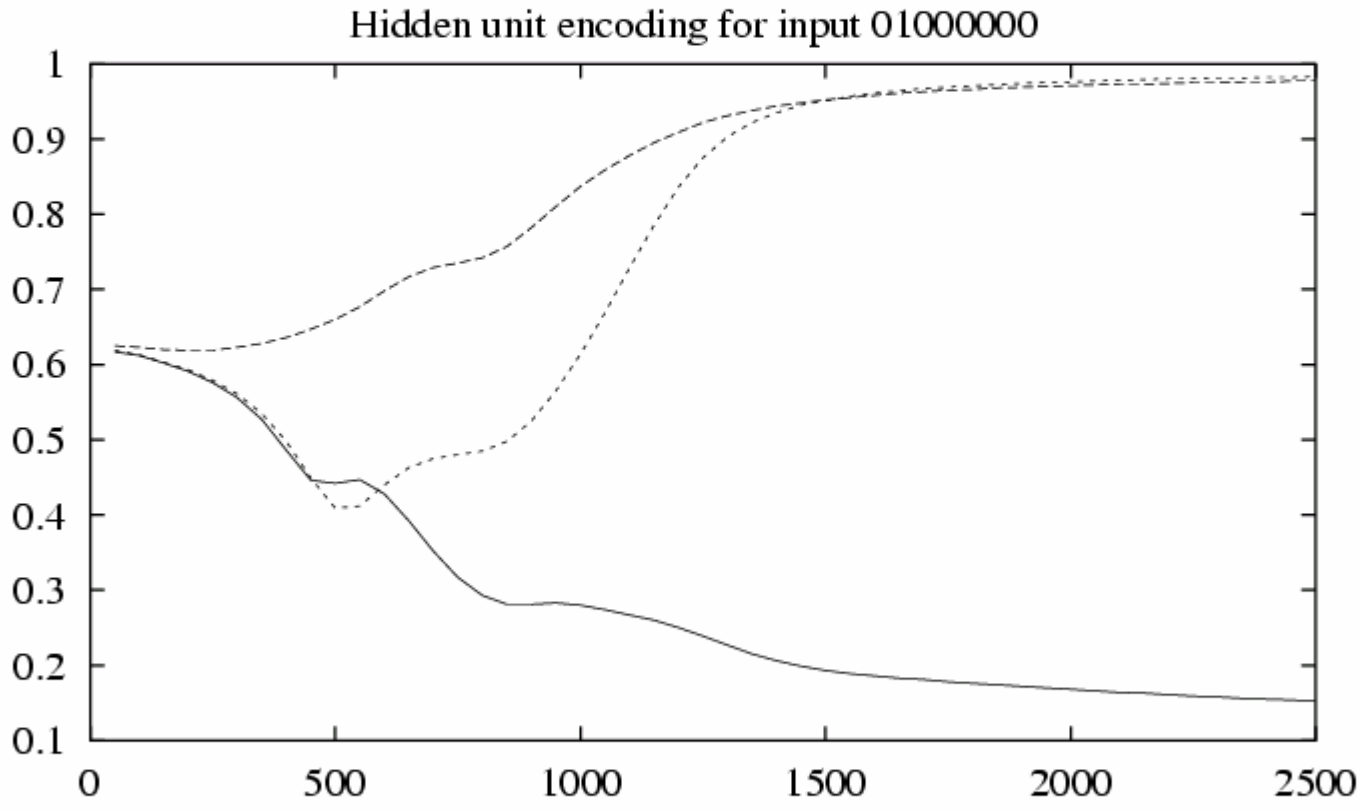
Learned hidden layer representation:

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

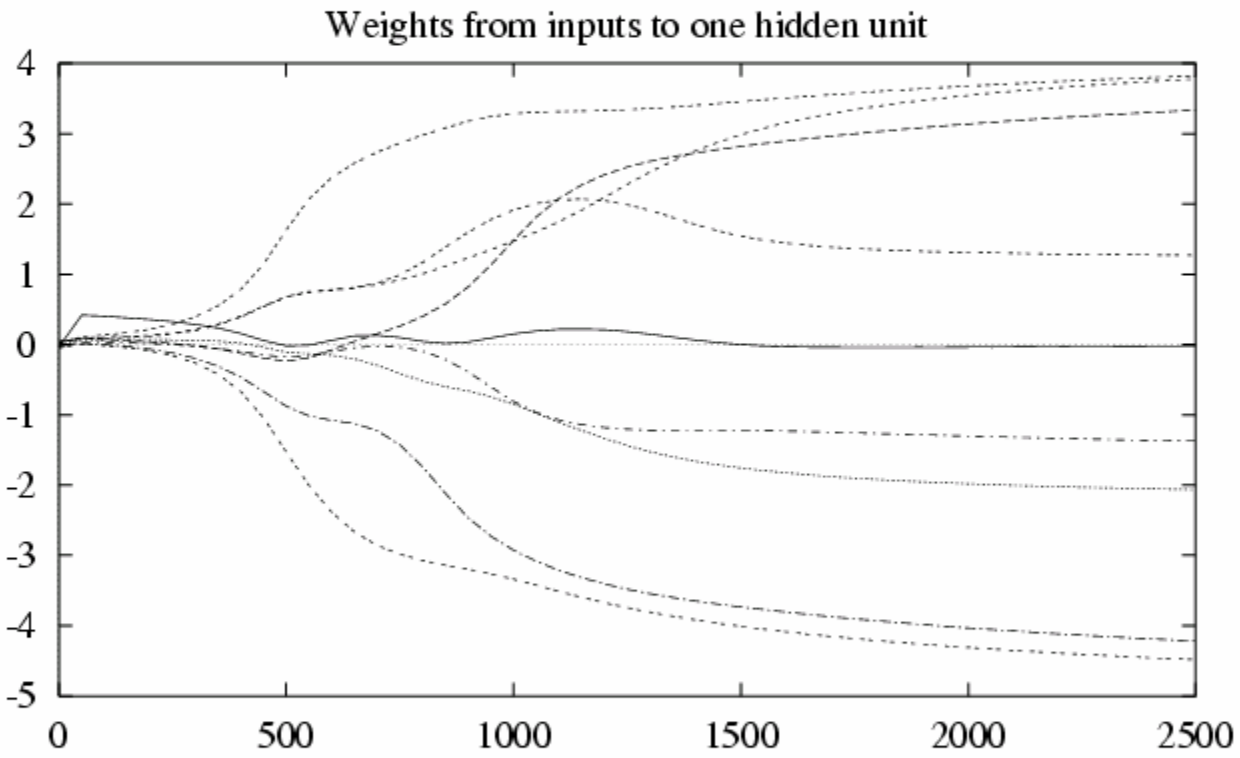
Training


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

Training


Hidden unit encoding for input 01000000
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
500
1000
1500
2000
2500

Training


Weights from inputs to one hidden unit
4
3
2
1
0
-1
-2
-3
-4
-5
0
500
1000
1500
2000
2500

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

Expressive Capabilities of ANNs

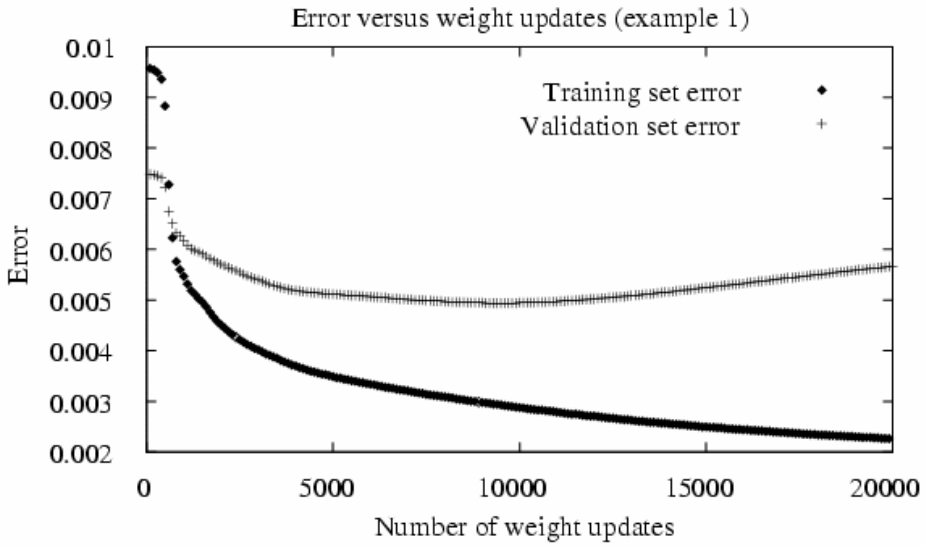
Boolean functions:

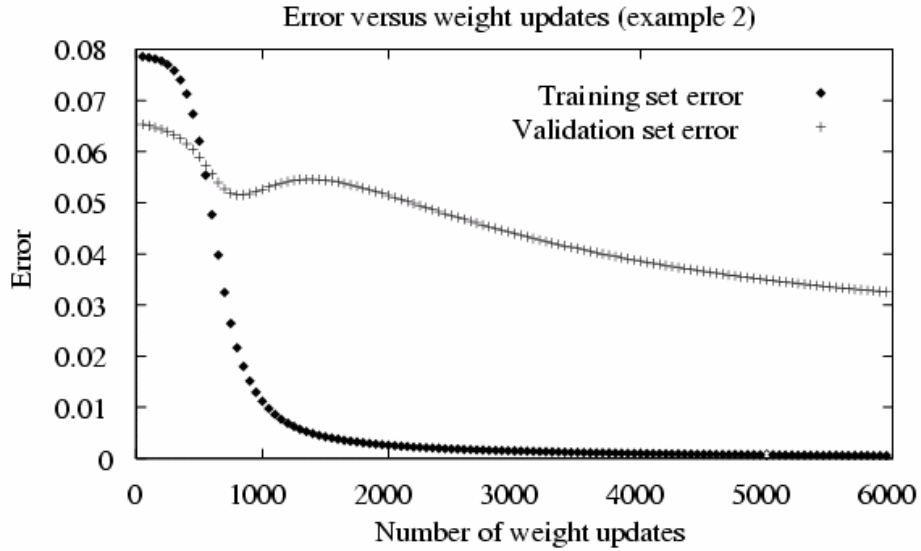
- Every boolean function can be represented by network with single hidden layer
- but might require exponential (in number of inputs) hidden units

Continuous functions:

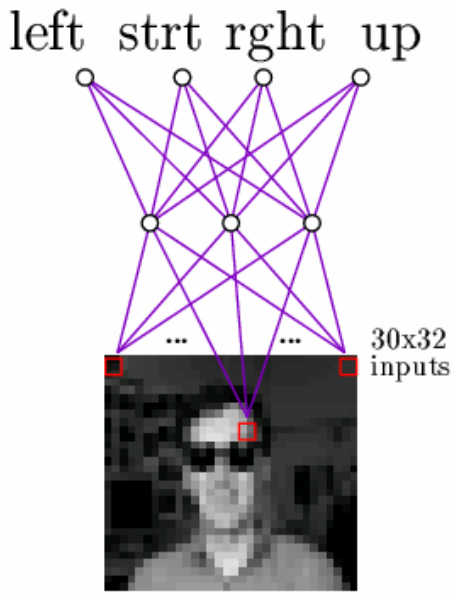
- Every bounded continuous function can be approximated with arbitrarily small error, by network with one hidden layer [Cybenko 1989; Hornik et al. 1989]
- Any function can be approximated to arbitrary accuracy by a network with two hidden layers [Cybenko 1988].

Overfitting in ANNs





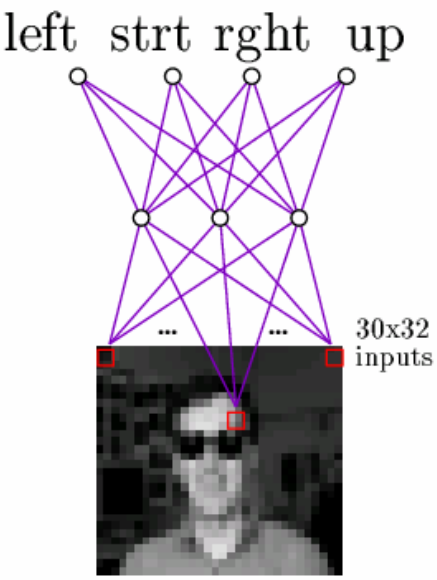
Neural Nets for Face Recognition



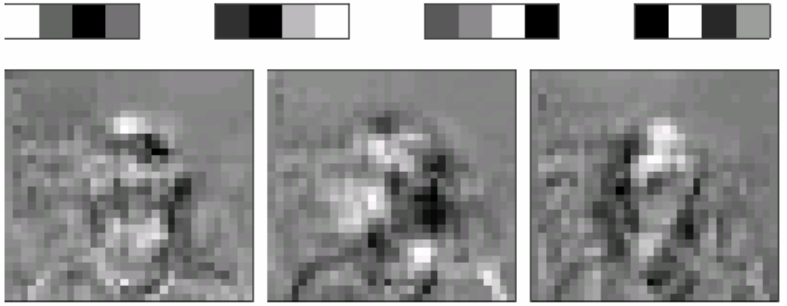
Typical input images

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

Learned Hidden Unit Weights



Learned Weights

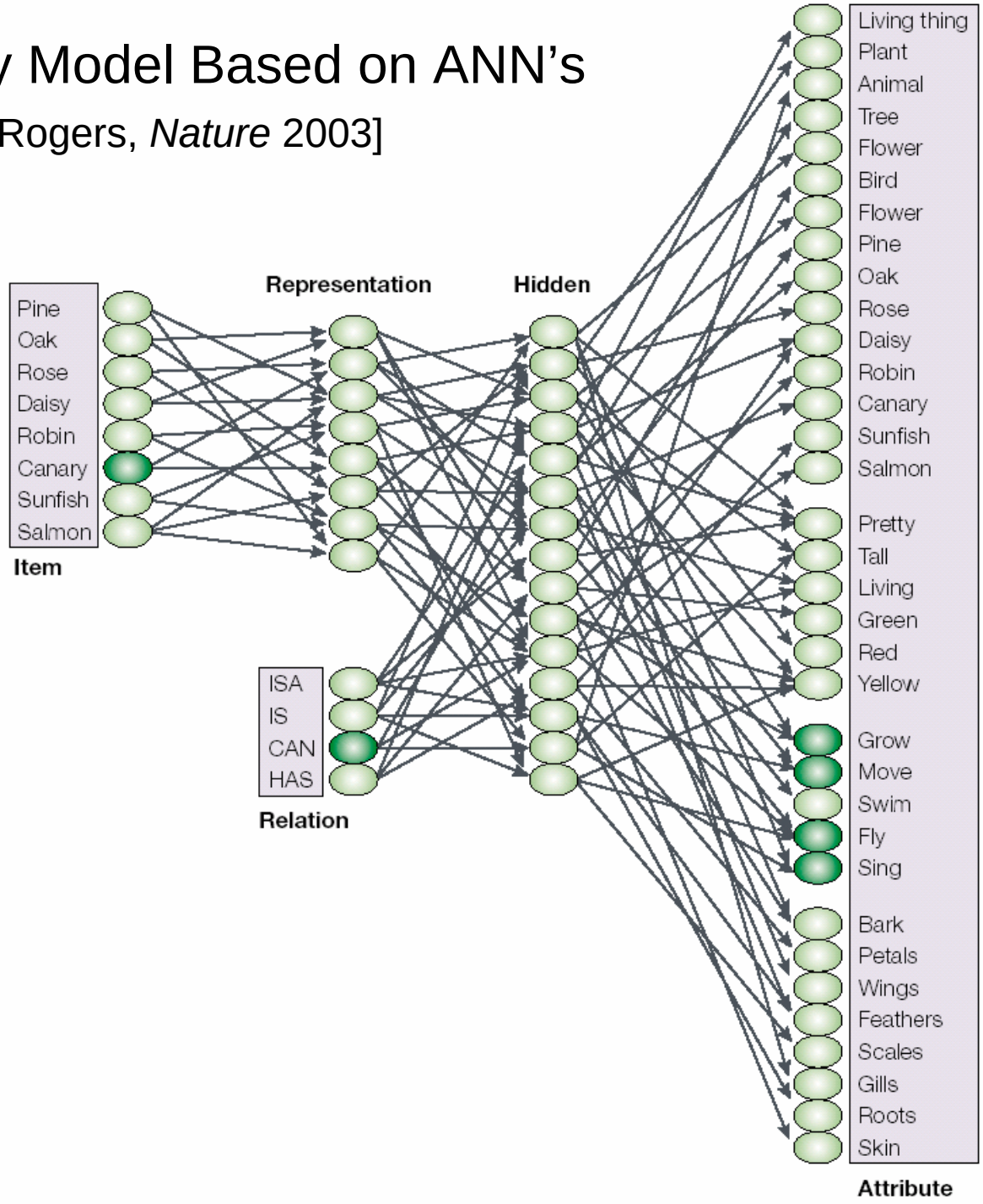


Typical input images

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

Semantic Memory Model Based on ANN's

[McClelland & Rogers, *Nature* 2003]



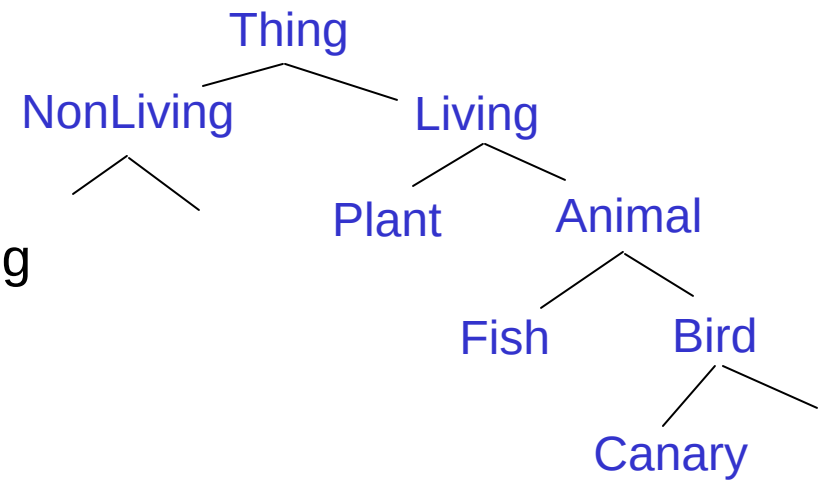
No hierarchy given.

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

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

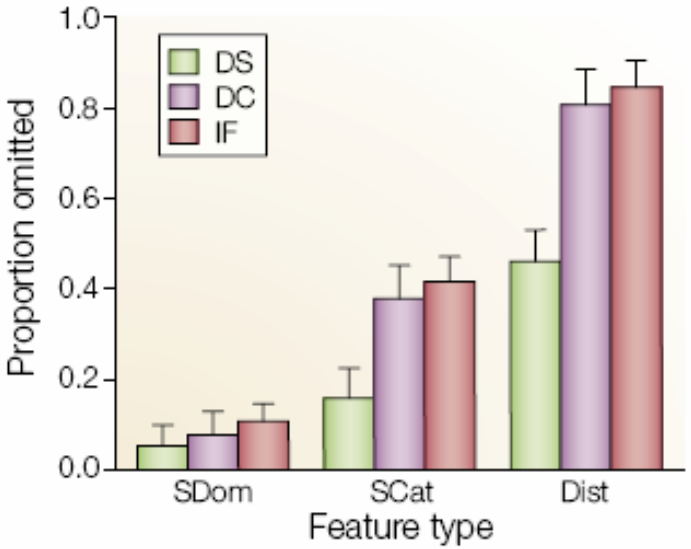
[McClelland & Rogers, *Nature* 2003]

a

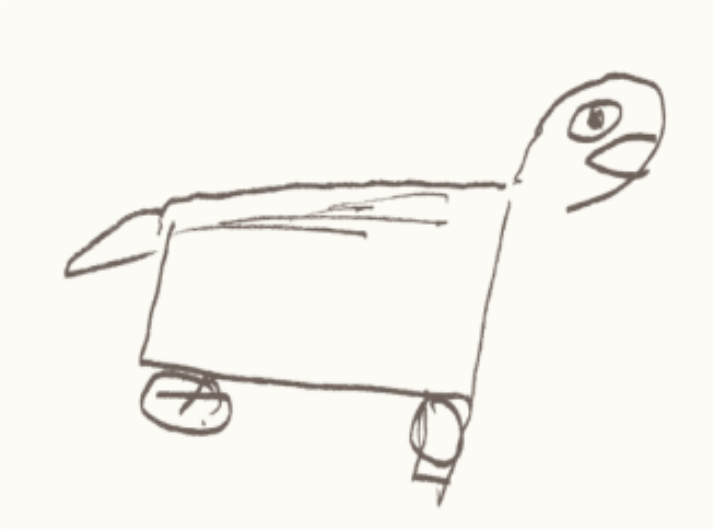
Picture naming responses for JL

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

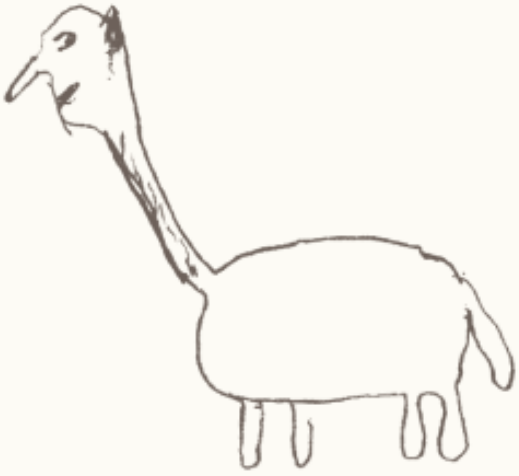
b



c IF's delayed copy of a camel



d DC's delayed copy of a swan



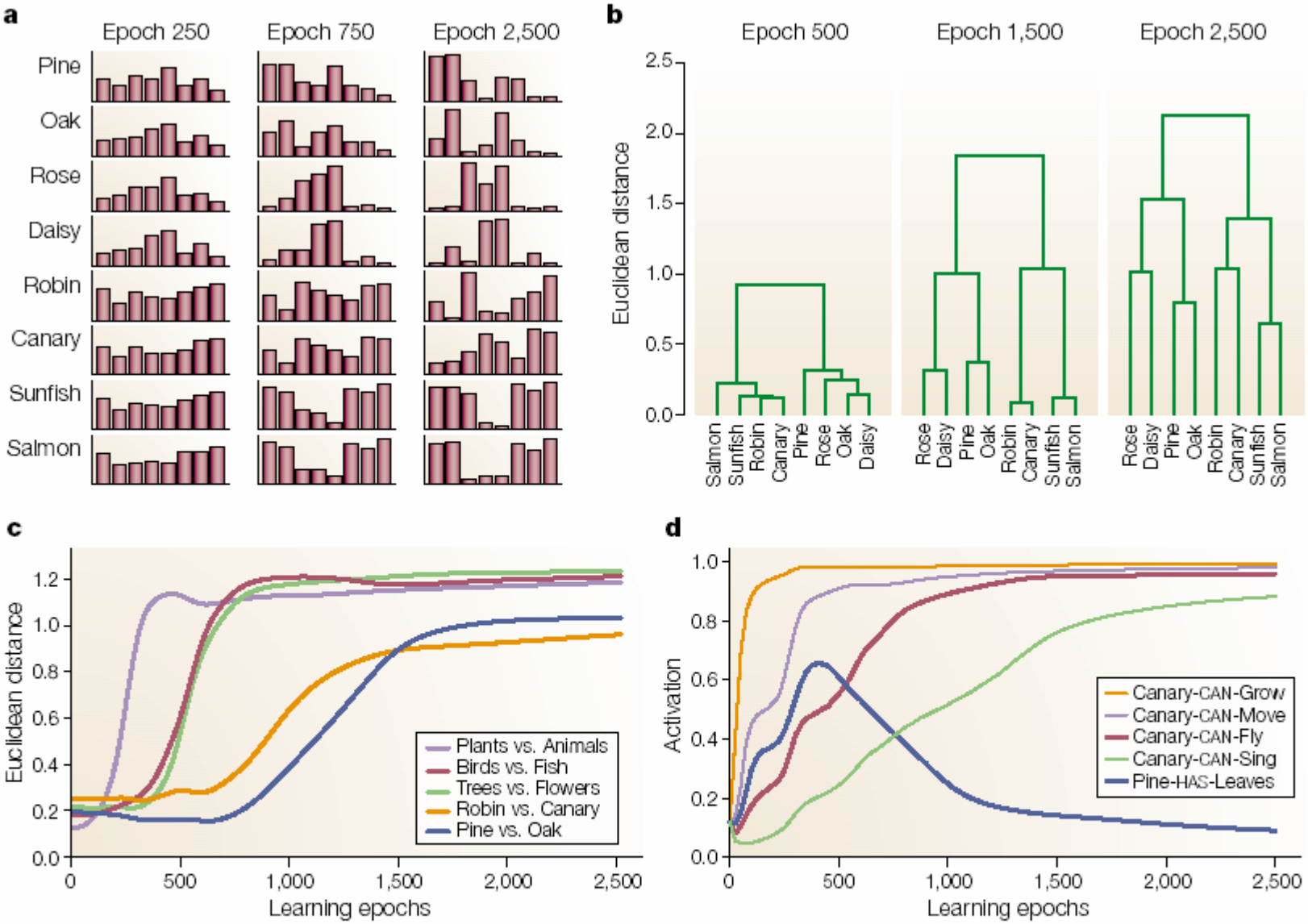
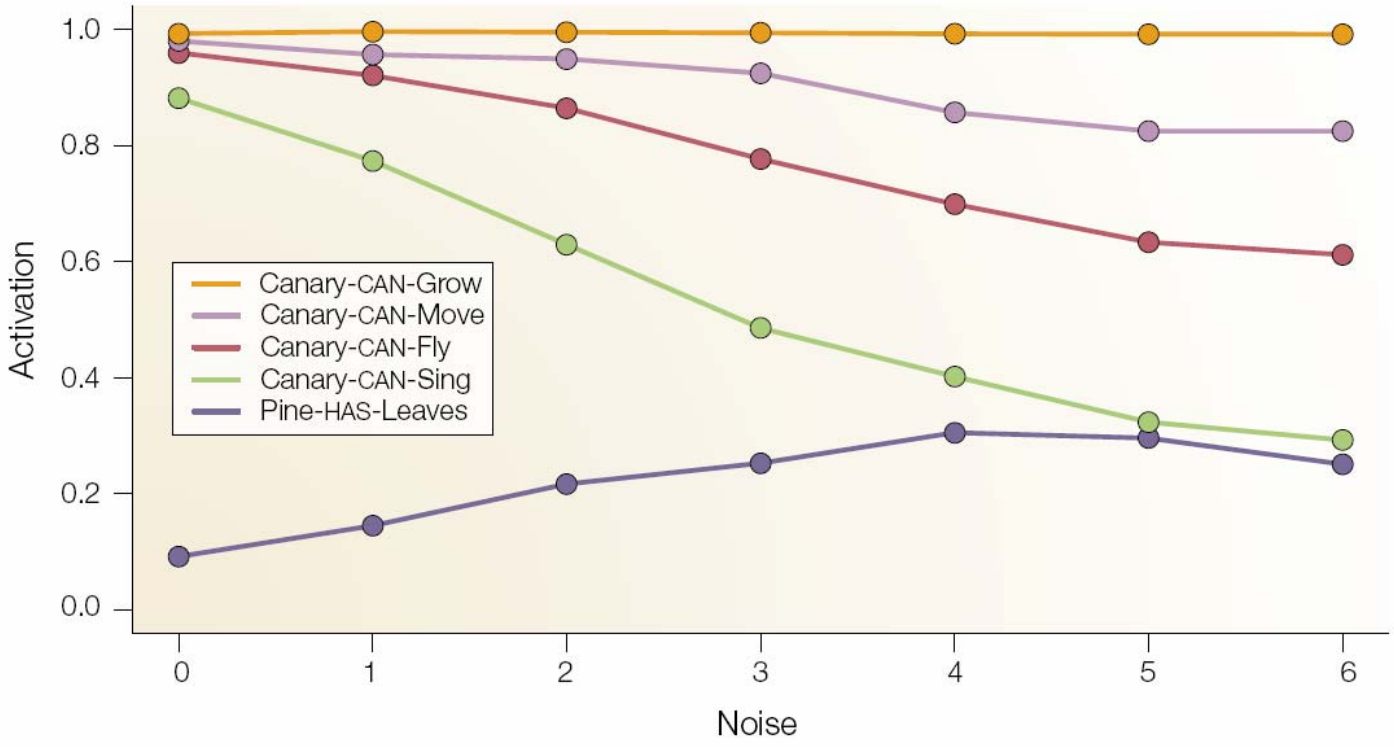


Figure 4 | **The process of differentiation of conceptual representations.** The representations are those seen in the feedforward network model shown in FIG. 3. **a** | Acquired patterns of activation that represent the eight objects in the training set at three points in the learning process (epochs 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

ANN Also Models Progressive Deterioration

[McClelland & Rogers, *Nature* 2003]



average effect of noise in inputs to hidden layers

Alternative Error Functions

Original MLE error fn.

Penalize large weights:

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} (t_{kd} - o_{kd})^2 + \gamma \sum_{i,j} w_{ji}^2$$

Train on target slopes as well as values:

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} \left[(t_{kd} - o_{kd})^2 + \mu \sum_{j \in inputs} \left(\frac{\partial t_{kd}}{\partial x_d^j} - \frac{\partial o_{kd}}{\partial x_d^j} \right)^2 \right]$$

Tie together weights:

- e.g., in phoneme recognition network

Bias/Variance Decomposition of Error

Bias – Variance decomposition of error

Reading: Bishop chapter 9.1, 9.2

- Consider simple regression problem f:X→Y

$y = f(x) + \varepsilon$

noise N(0,σ)

deterministic

What are sources of prediction error?

$$E_D\left[\int_y \int_x (h(x) - f(x))^2 p(y|x) p(x) dy dx\right]$$

learned

Sources of error

- What if we have perfect learner, infinite data?
  - Our learned h(x) satisfies h(x)=f(x)
  - Still have remaining, *unavoidable error*

$$\sigma^2$$

Sources of error

- What if we have only n training examples?
- What is our expected error
  - Taken over random training sets of size n, drawn from distribution $D=p(x,y)$

$$E_D\left[\int_y\int_x (h(x)-f(x))^2 p(y|x)p(x)dydx\right]$$

Sources of error

$$E_D\left[\int_y\int_x (h(x)-f(x))^2 p(y|x)p(x)dydx\right]$$

$$= unavoidableError + bias^2 + variance$$

$$bias^2 = \int (E_D[h(x)]-f(x))^2 p(x)dx$$

$$variance = \int E_D[(h(x)-E_D[h(x)])^2]p(x)dx$$