



# Artificial Neural Networks

Reading:

- Neural nets: Mitchell chapter 4

Machine Learning 10-601

Tom M. Mitchell  
Machine Learning Department  
Carnegie Mellon University

April 9, 2008

# Artificial Neural Networks to learn $f: X \rightarrow Y$

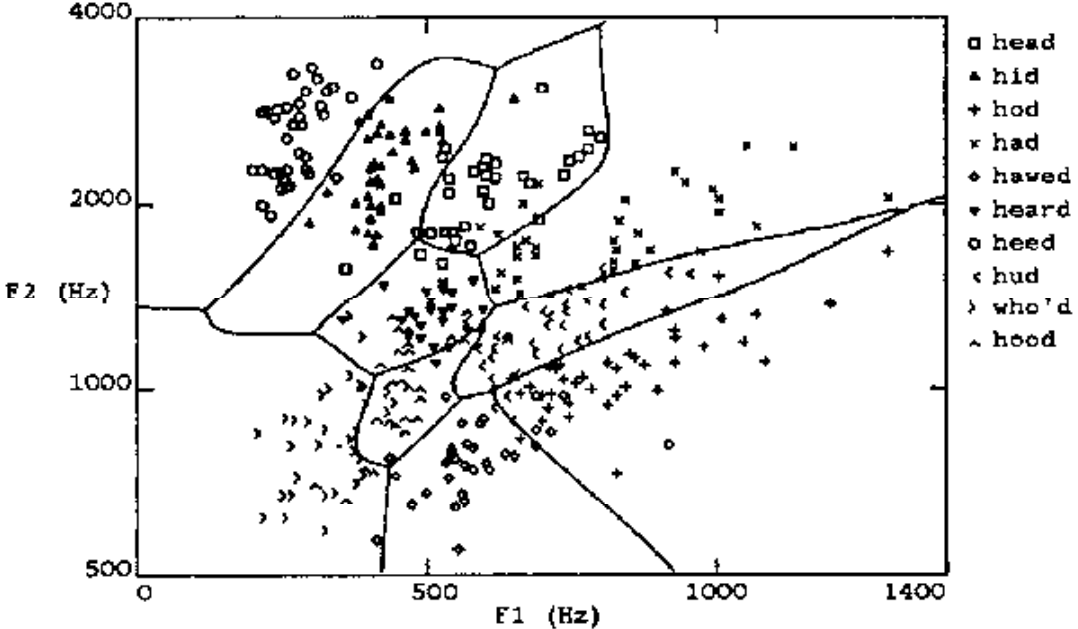
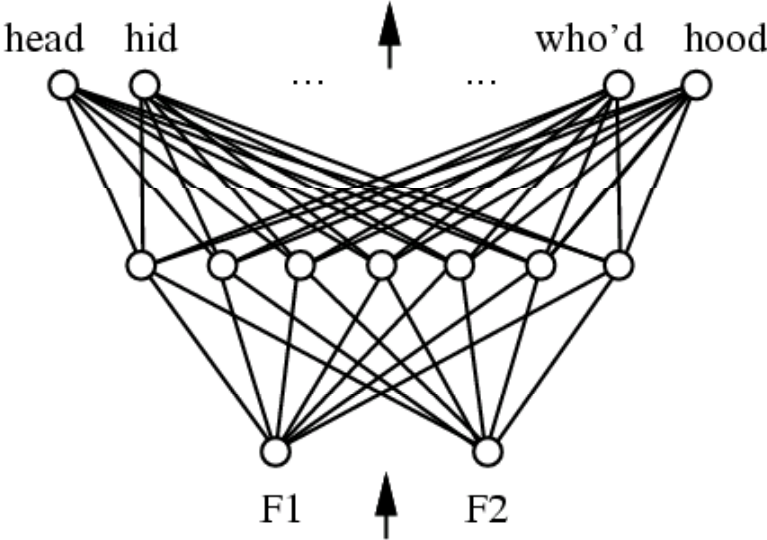
- $f$  might be non-linear function
- $X$  (vector of) continuous and/or discrete vars
- $Y$  (vector of) continuous and/or discrete vars

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

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

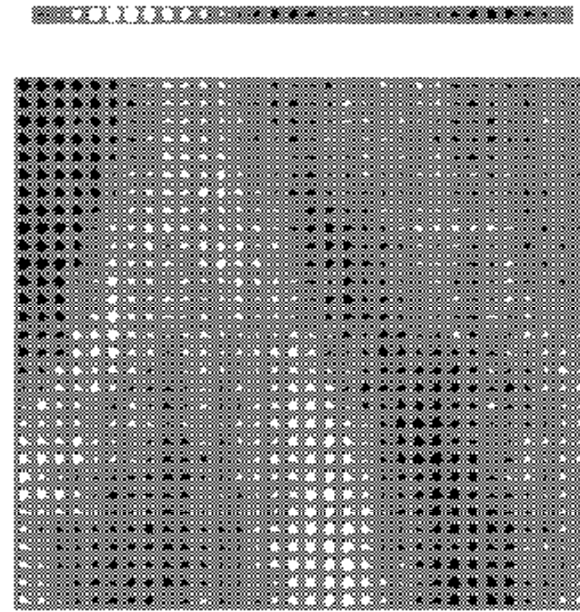
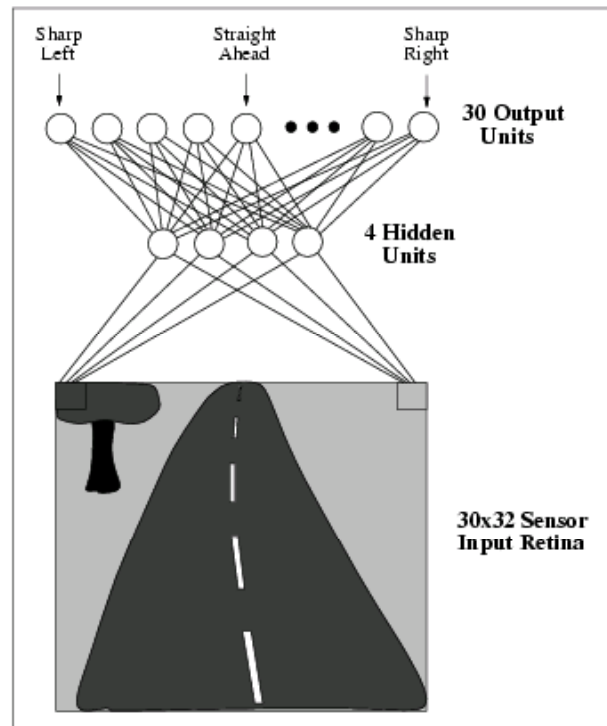
- MLE: train weights of all units to minimize sum of squared errors of predicted network outputs

# Multilayer Networks of Sigmoid Units



# ALVINN

[Pomerleau 1993]



# Connectionist Models

---

Consider humans:

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

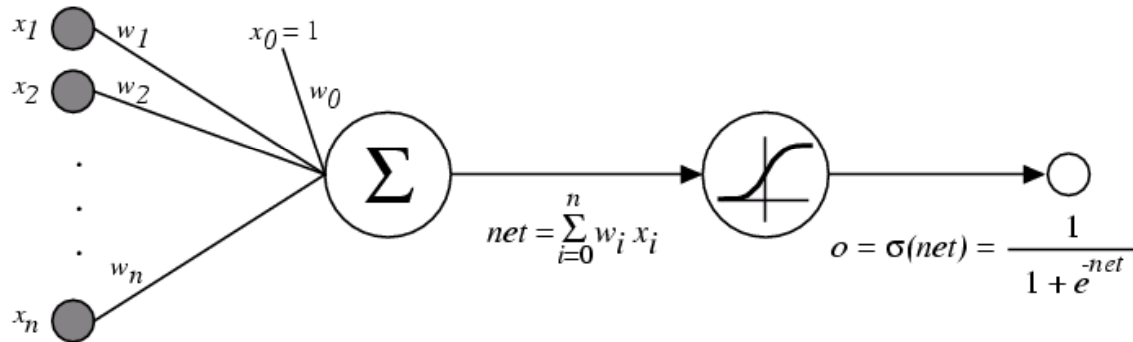
→ much parallel computation

Properties of artificial neural nets (ANN's):

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

# Sigmoid Unit

---



$\sigma(x)$  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  $f: X \rightarrow Y$ , for scalar  $Y$

$$y = f(x) + \varepsilon \quad \leftarrow \quad \text{assume noise } N(0, \sigma_\varepsilon), \text{ iid}$$

deterministic

- Let's maximize the conditional data likelihood

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

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

Learned  
neural network

# MAP Training for Neural Networks

- Consider regression problem  $f: X \rightarrow Y$ , for scalar  $Y$

$$y = f(x) + \varepsilon \quad \leftarrow \text{noise } N(0, \sigma_\varepsilon)$$

deterministic

Gaussian  $P(W) = N(0, \sigma I)$

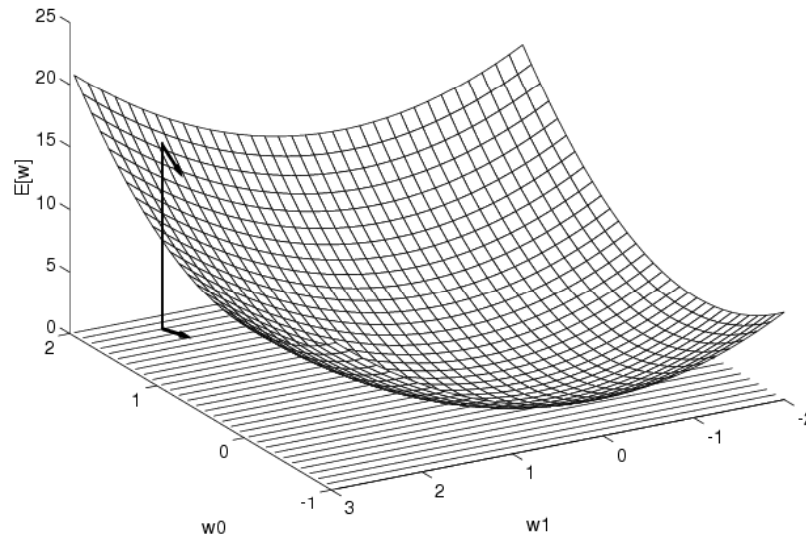
$$W \leftarrow \arg \max_W \ln P(W) \prod_l P(Y^l | X^l, W)$$

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

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

# Gradient Descent

---



Gradient

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

Training rule:

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

i.e.,

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

# Error Gradient for a Sigmoid Unit

---

$x_d$  = input

$t_d$  = target output

$o_d$  = observed unit output

$w_i$  = weight  $i$

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

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  $\eta$  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  $h$

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

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  $j$

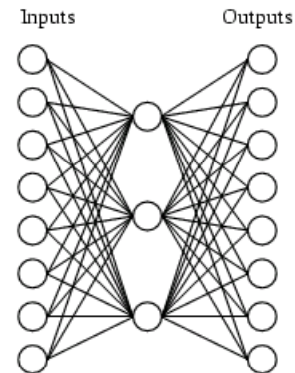
# More on Backpropagation

---

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

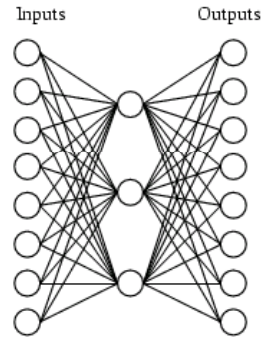
Input	Output
10000000	→ 10000000
01000000	→ 01000000
00100000	→ 00100000
00010000	→ 00010000
00001000	→ 00001000
00000100	→ 00000100
00000010	→ 00000010
00000001	→ 00000001

Can this be learned??

# Learning Hidden Layer Representations

---

A network:

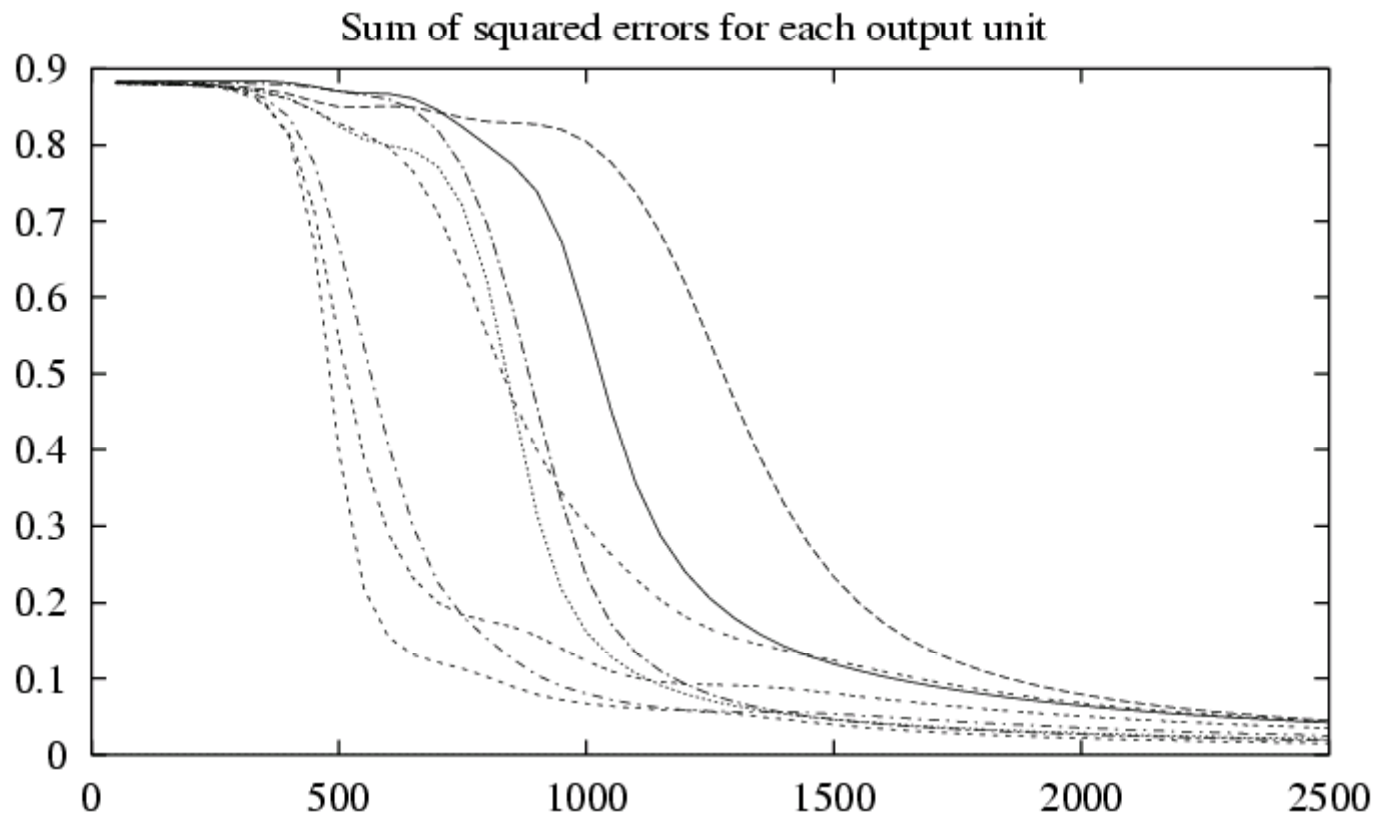


Learned hidden layer representation:

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

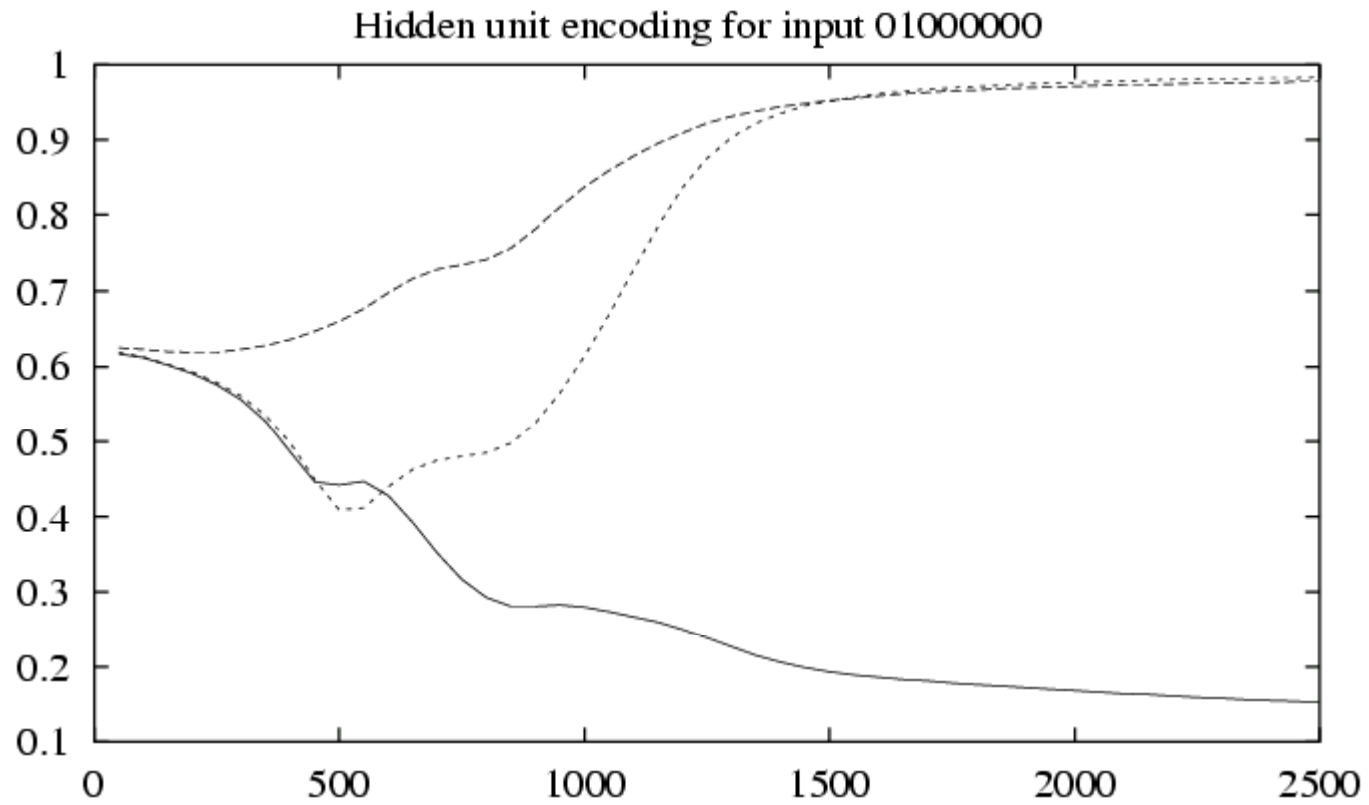
# Training

---



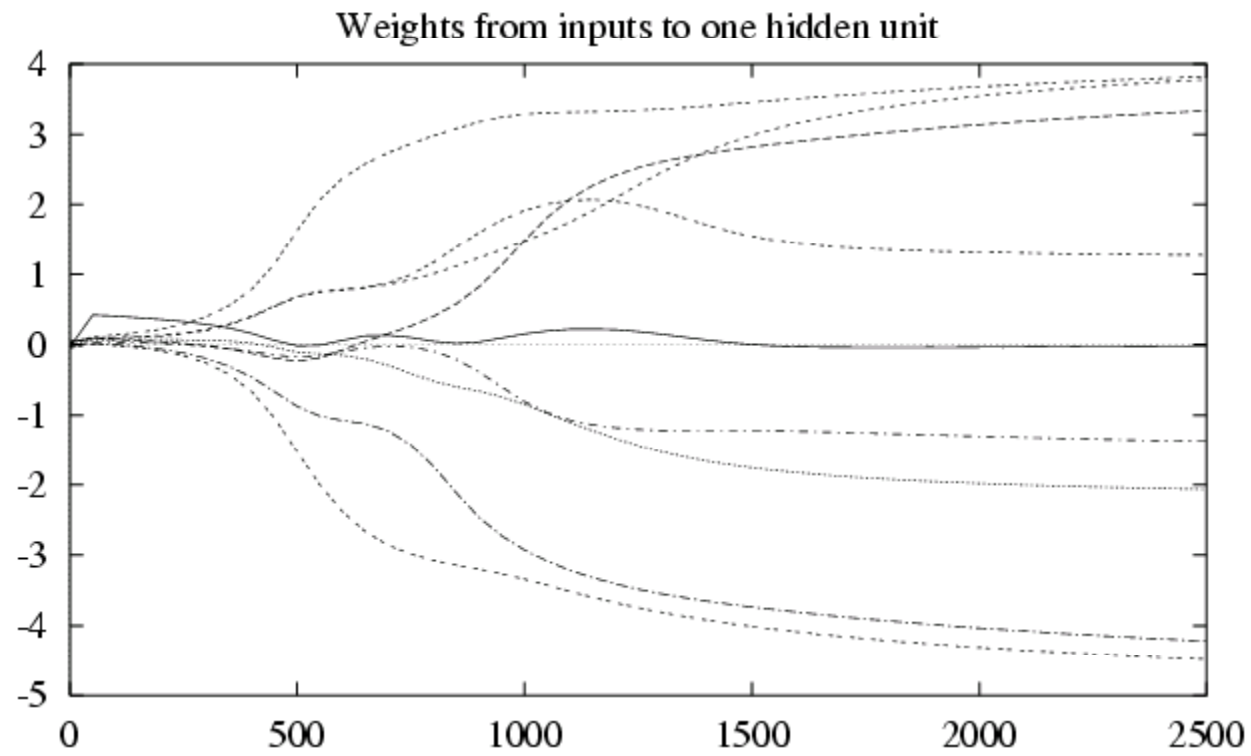
# Training

---



# Training

---



# Convergence of Backpropagation

---

Gradient descent to some local minimum

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

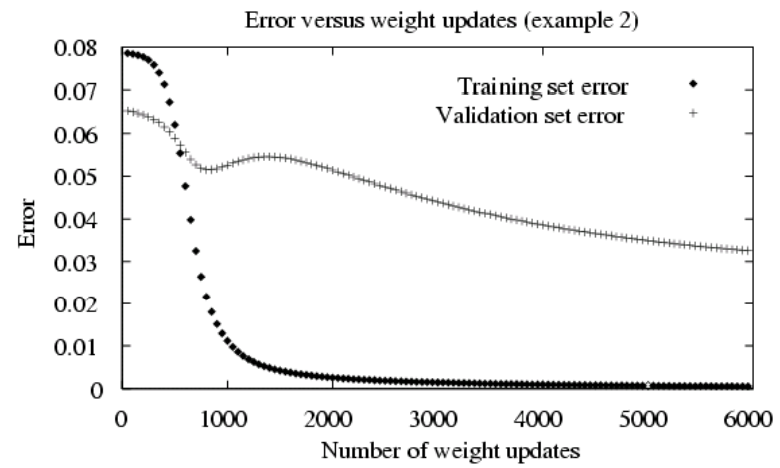
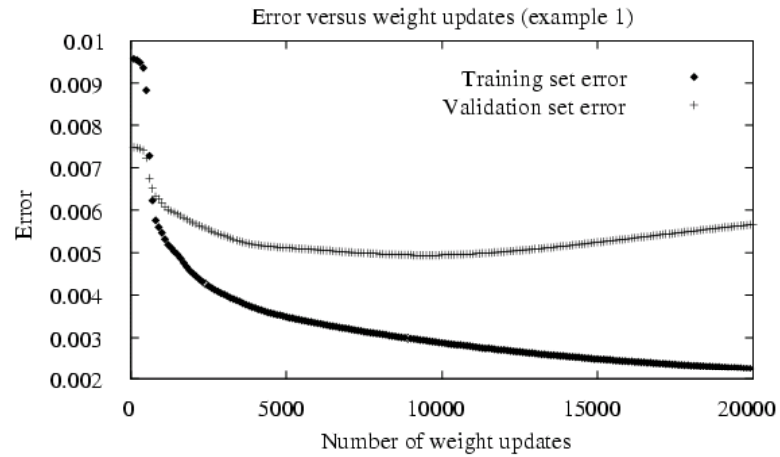
- Every boolean function can be represented by network with single hidden layer
- but might require exponential (in number of inputs) hidden units

## 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].

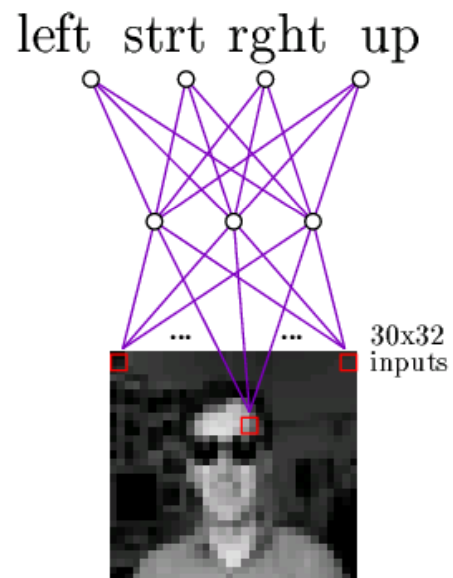
# Overfitting in ANNs

---



# Neural Nets for Face Recognition

---

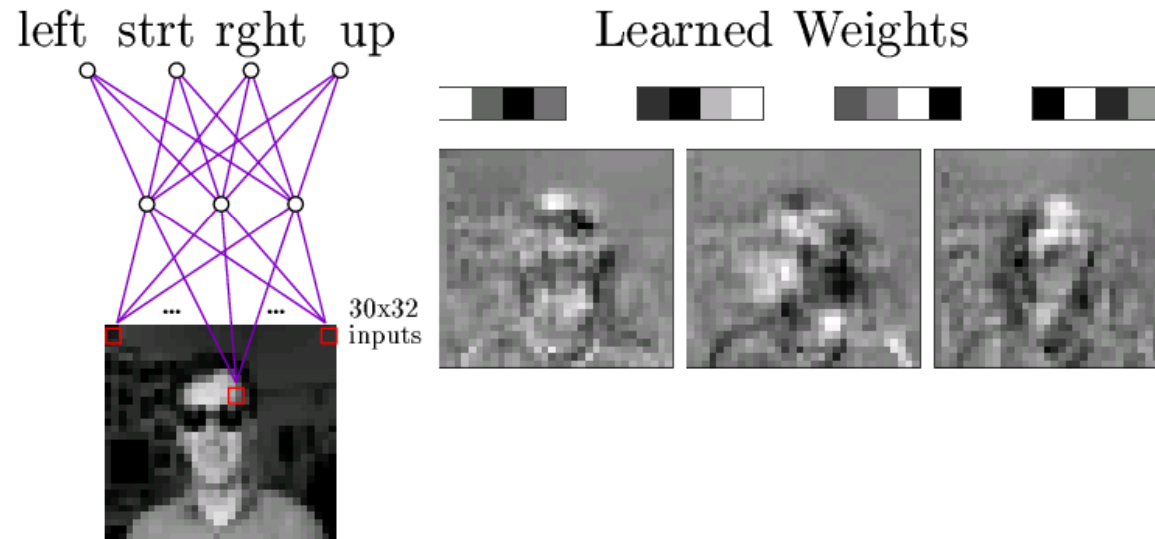


Typical input images

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

# Learned Hidden Unit Weights

---



Typical input images

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

# What you should know

- Feature selection
  - Single feature scoring criteria
  - Regularization
  - Search strategies
    - Common approaches: Greedy addition of features, or greedy deletion
- Unsupervised dimension reduction using all features
  - Principle Components Analysis
    - Minimize reconstruction error
  - Singular Value Decomposition
    - Efficient PCA
  - Independent components analysis
- Supervised dimension reduction
  - Fisher Linear Discriminant
    - Project to  $n-1$  dimensions to discriminate  $n$  classes
  - Hidden layers of Neural Networks
    - Most flexible, local minima issues

# Further Readings

- “Singular value decomposition and principal component analysis,” Wall, M.E, Rechtsteiner, A., and L. Rocha, in *A Practical Approach to Microarray Data Analysis* (D.P. Berrar, W. Dubitzky, M. Granzow, eds.) Kluwer, Norwell, MA, 2003. pp. 91-109.

LANL LA-UR-02-4001