Neural Networks

Required reading:

• Bishop Chapter 5, especially 5.1, 5.2, 5.3, and 5.5 through 5.5.2

Optional reading:

• Neural nets: Mitchell chapter 4

Machine Learning 10-701

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

September 28, 2006

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 of network function

Connectionist Models

Consider humans:

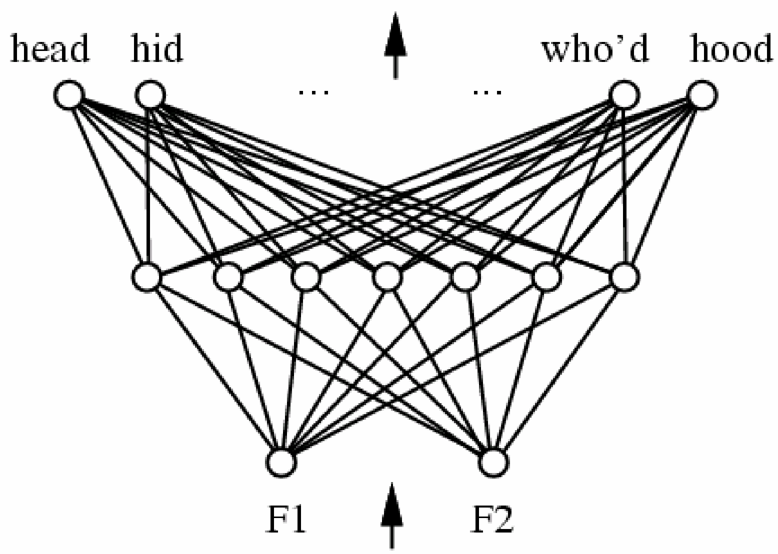
- Neuron switching time ~ .001 second
- Number of neurons ~ 10^{10}
- Connections per neuron ~ 10^{4-5}
- Scene recognition time ~ .1 second
- 100 inference steps doesn't seem like enough

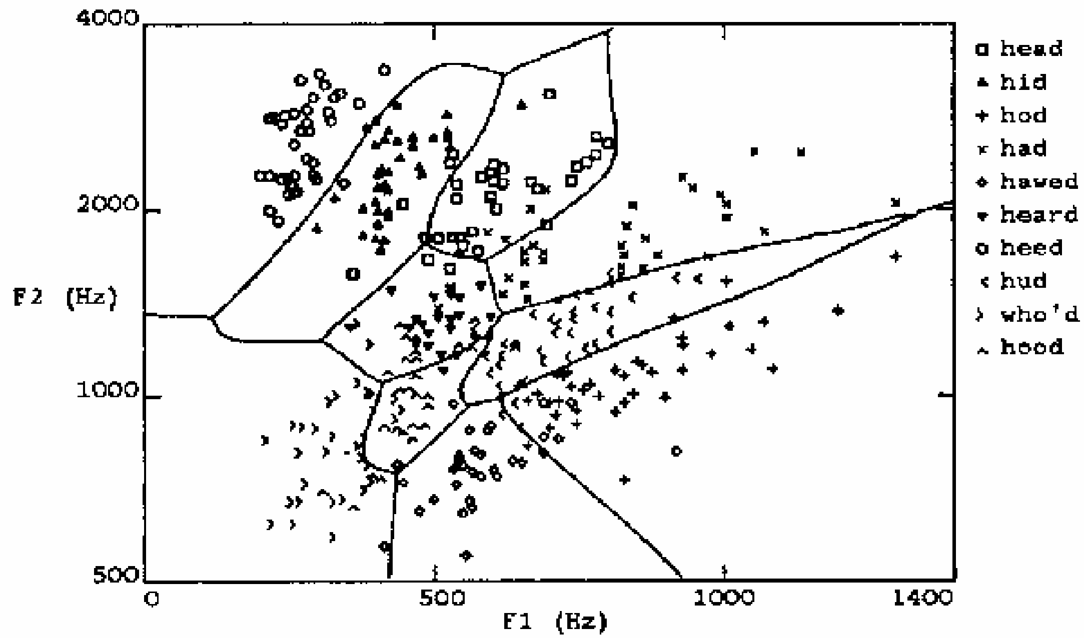
$\rightarrow$ much parallel computation

Properties of artificial neural nets (ANN's):

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

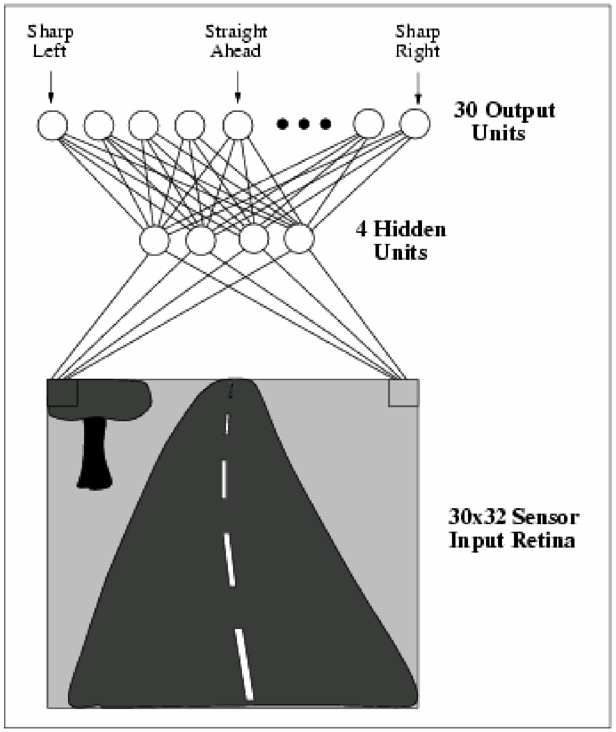
Multilayer Networks of Sigmoid Units

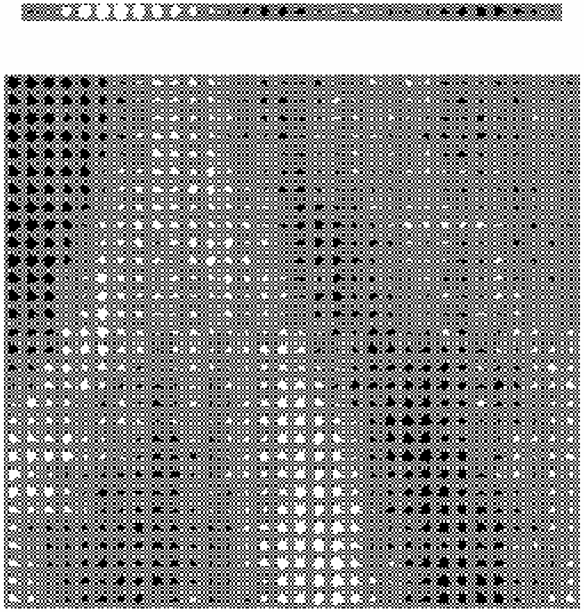




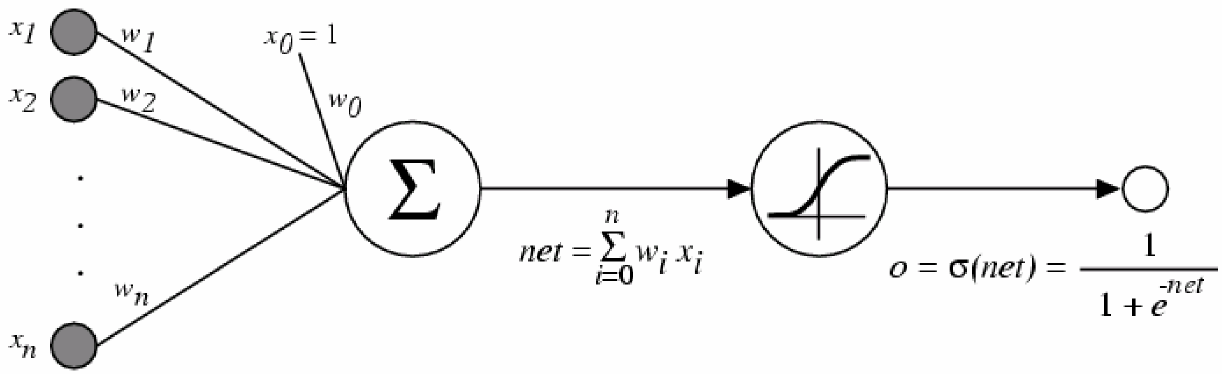
ALVINN

[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

M(C)LE Training for Neural Networks

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

$y = f(x) + \varepsilon$ ← 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

$$y = f(x) + \varepsilon$$

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

deterministic

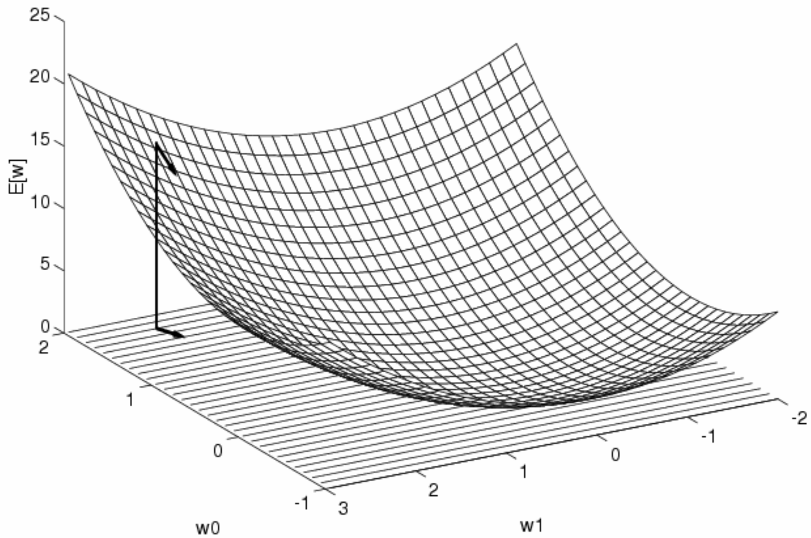
Gaussian $P(W) = N(0,\sigma I)$

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

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

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

Gradient Descent



Gradient

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

Training rule:

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

i.e.,

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

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 η made small enough

Backpropagation Algorithm

x_d = input

t_d = target output

o_d = observed unit output

w_{ij} = wt from i to j

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

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

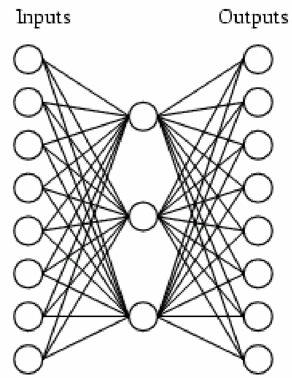
More on Backpropagation

- 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



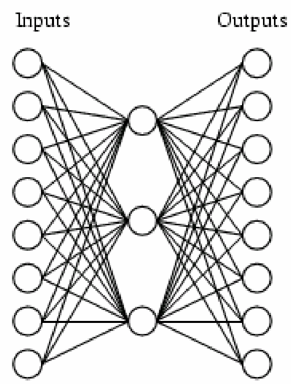
A target function:

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

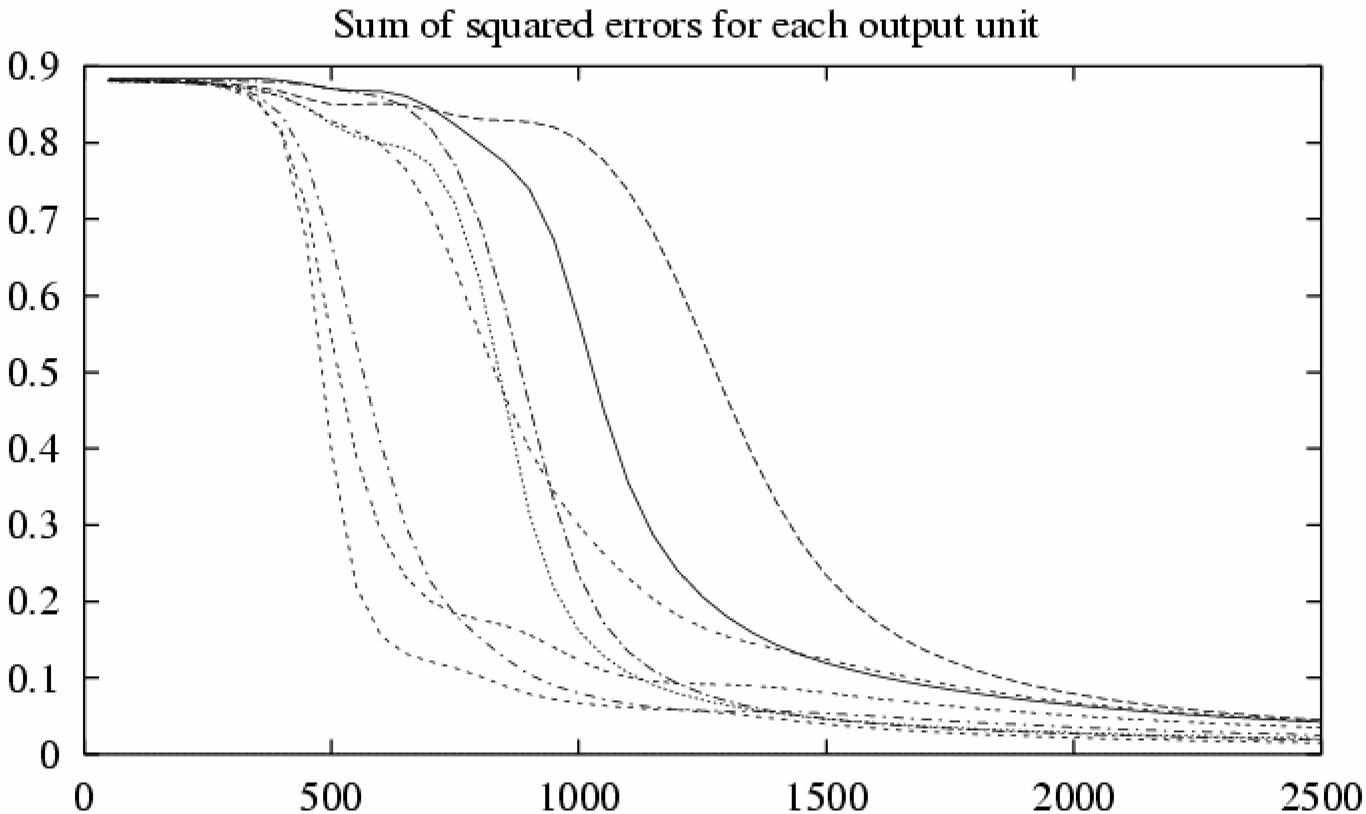
A network:



Learned hidden layer representation:

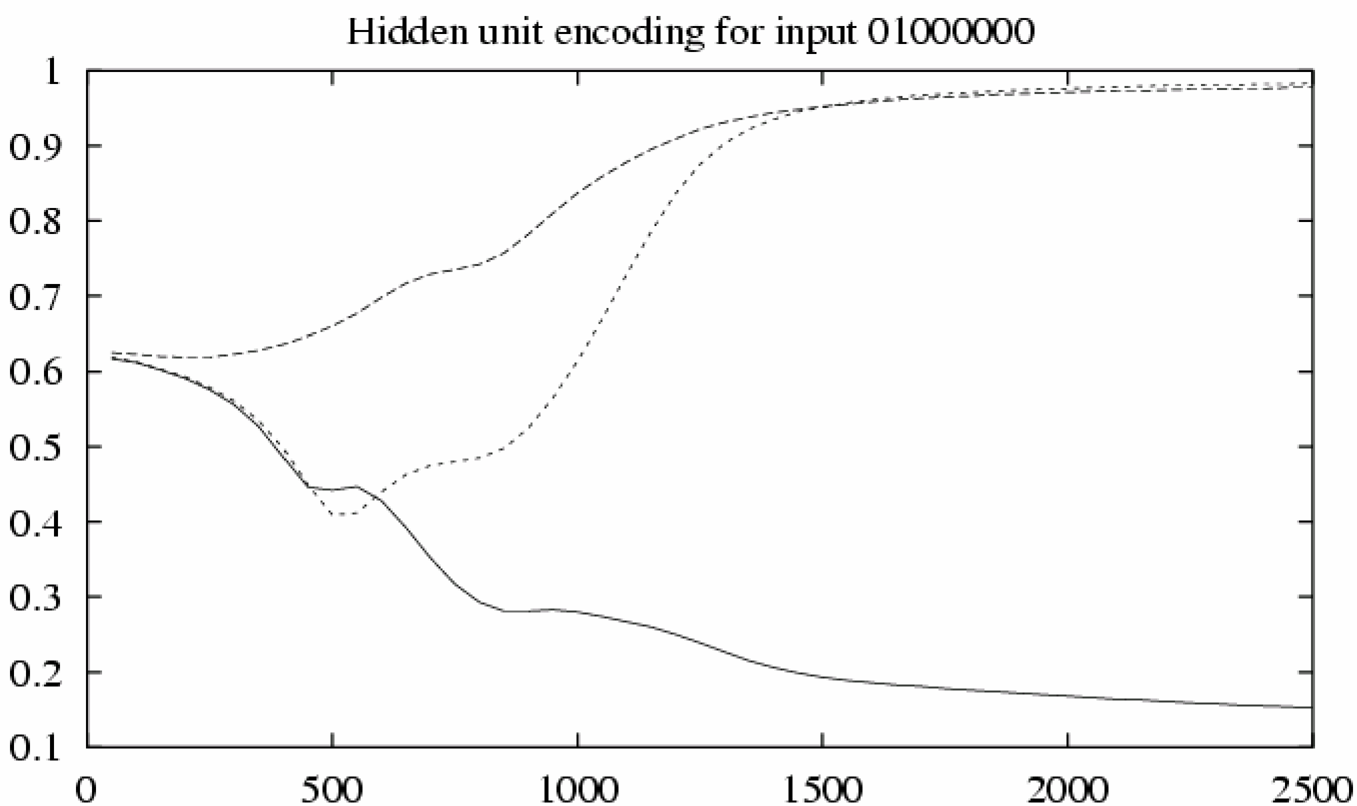
Input		Hidden Values				Output
10000000	$\rightarrow$	.89	.04	.08	$\rightarrow$	10000000
01000000	$\rightarrow$	.01	.11	.88	$\rightarrow$	01000000
00100000	$\rightarrow$	.01	.97	.27	$\rightarrow$	00100000
00010000	$\rightarrow$	.99	.97	.71	$\rightarrow$	00010000
00001000	$\rightarrow$	.03	.05	.02	$\rightarrow$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

Training



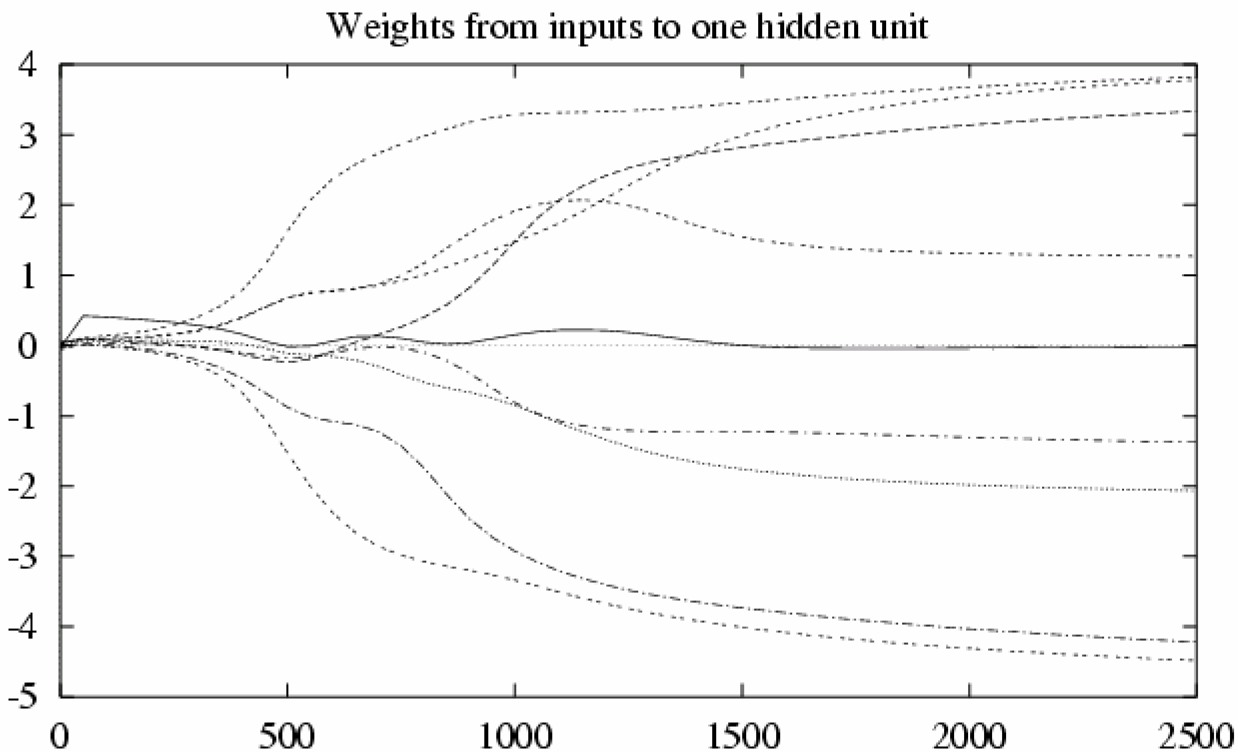
Sum of squared errors for each output unit
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
500
1000
1500
2000
2500

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

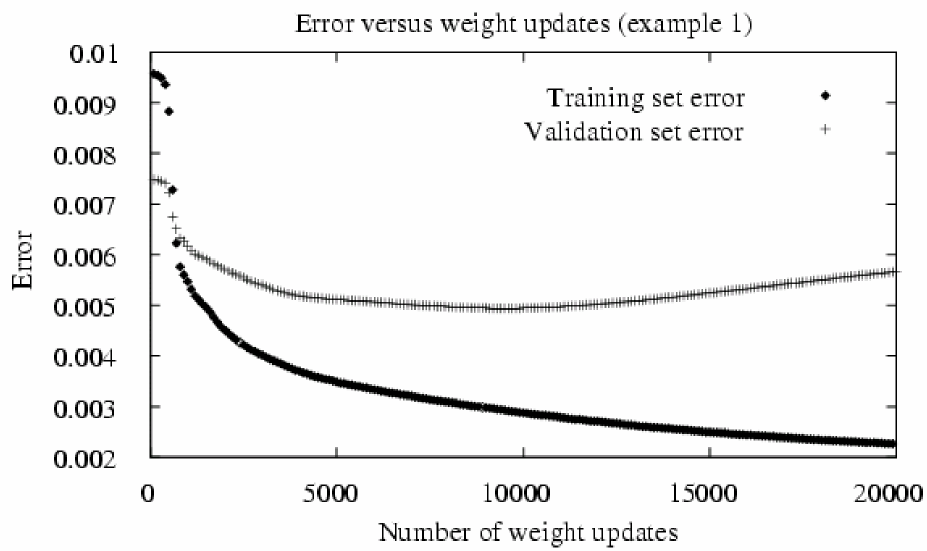
Boolean functions:

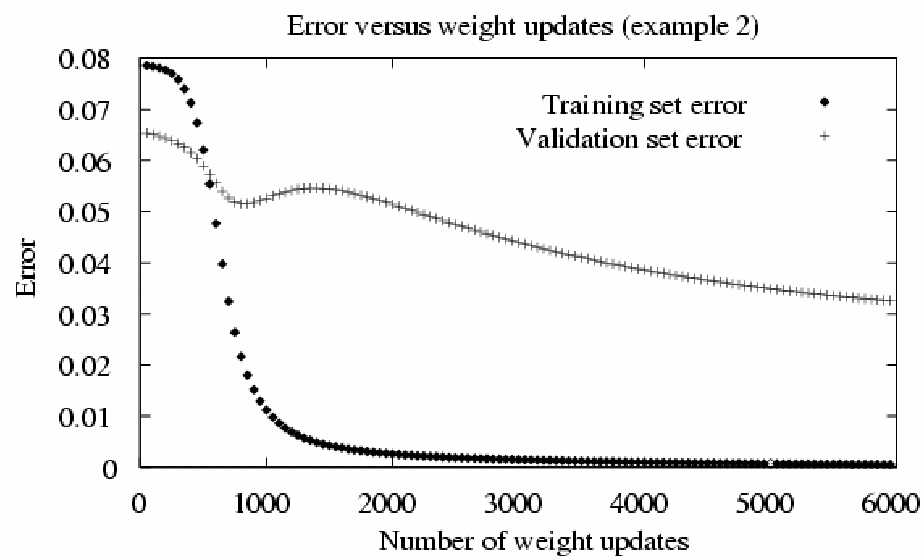
- Every boolean function can be represented by network with single hidden layer
- but might require exponential (in number of inputs) hidden units

Continuous functions:

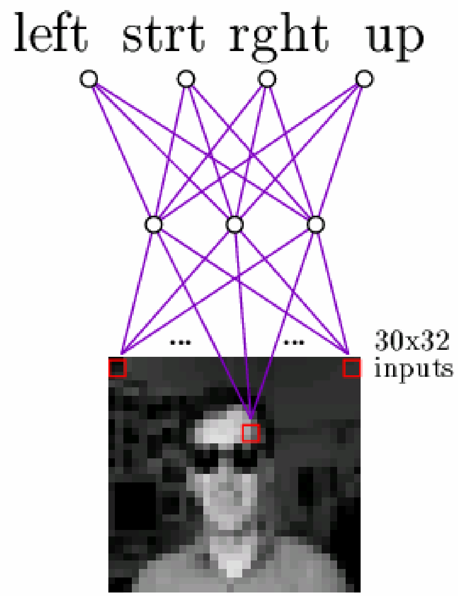
- Every bounded continuous function can be approximated with arbitrarily small error, by network with one hidden layer [Cybenko 1989; Hornik et al. 1989]
- Any function can be approximated to arbitrary accuracy by a network with two hidden layers [Cybenko 1988].

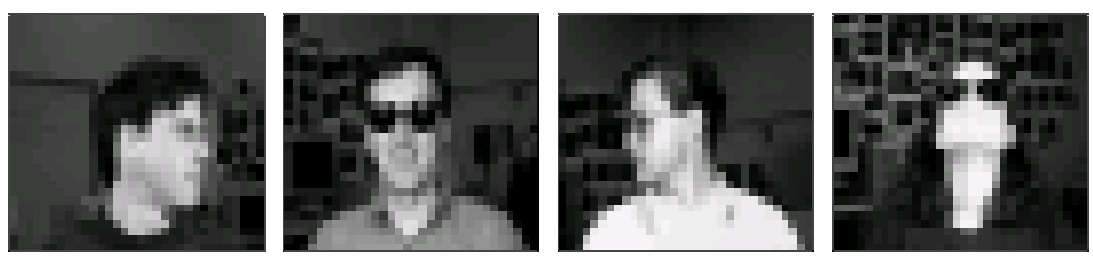
Overfitting in ANNs





Neural Nets for Face Recognition

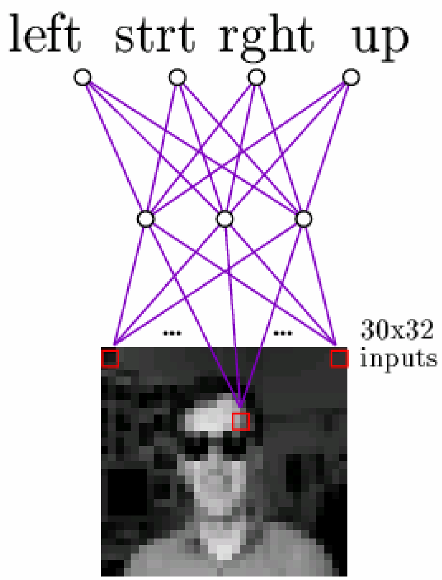




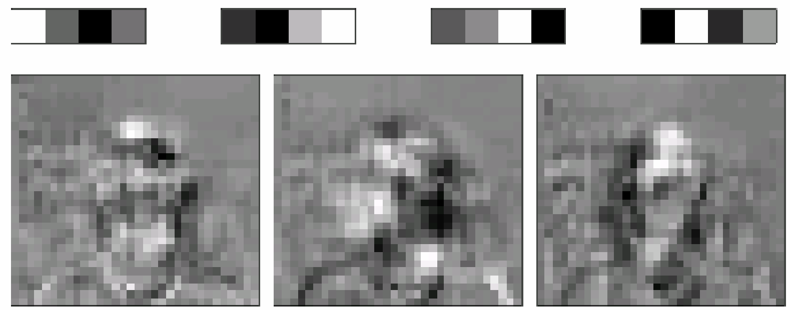
Typical input images

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

Alternative Error Functions

Original MLE error fn.

Penalize large weights:

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

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

Artificial neural networks – what you should know

- Highly expressive non-linear functions
- Highly parallel network of logistic function units
- Minimizing sum of squared training errors
  - Gives MLE estimates of network weights if we assume zero mean Gaussian noise on output values
- Minimizing sum of sq errors plus weight squared (regularization)
  - MAP estimates assuming weight priors are zero mean Gaussian
- Gradient descent as training procedure
  - How to derive your own gradient descent procedure
- Discover useful representations at hidden units
- Local minima is greatest problem
- Overfitting, regularization, early stopping