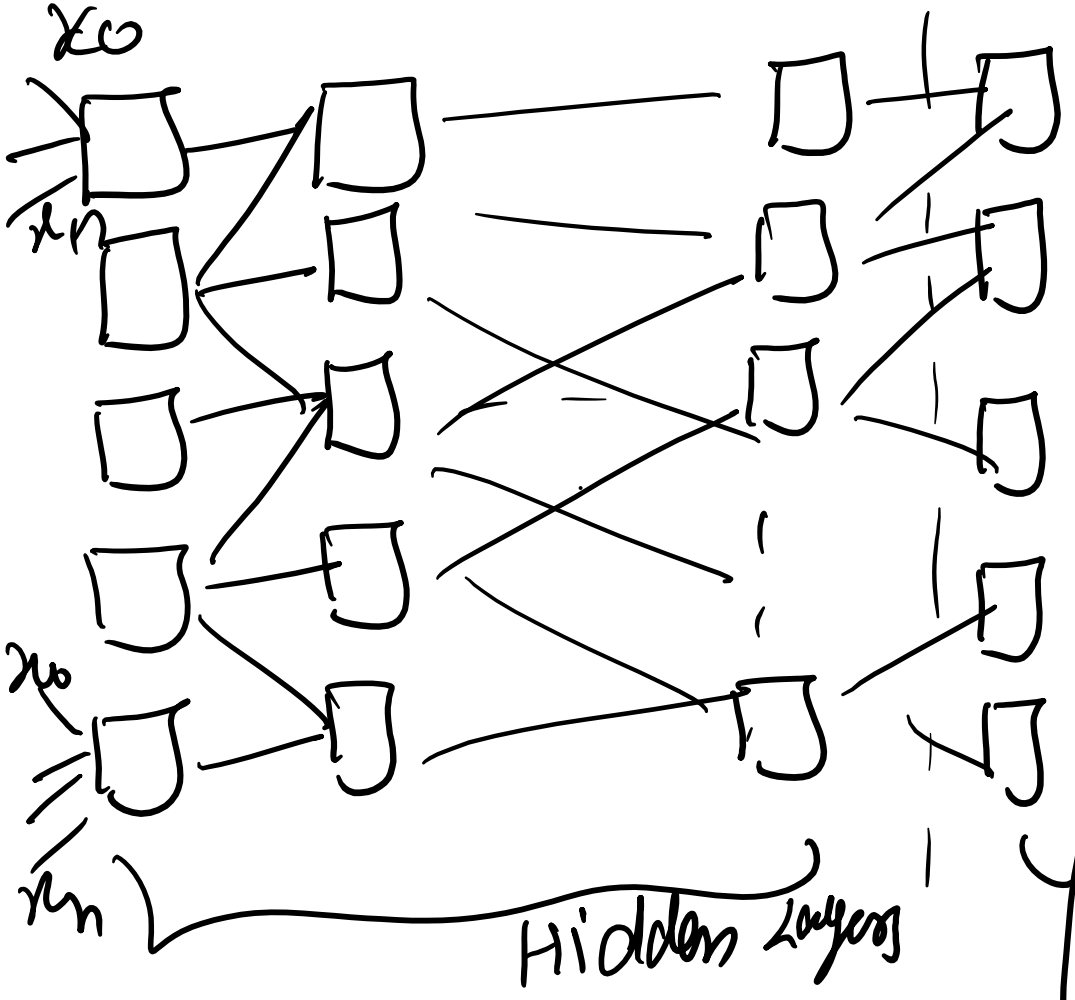
Machine Learning (COL 774)

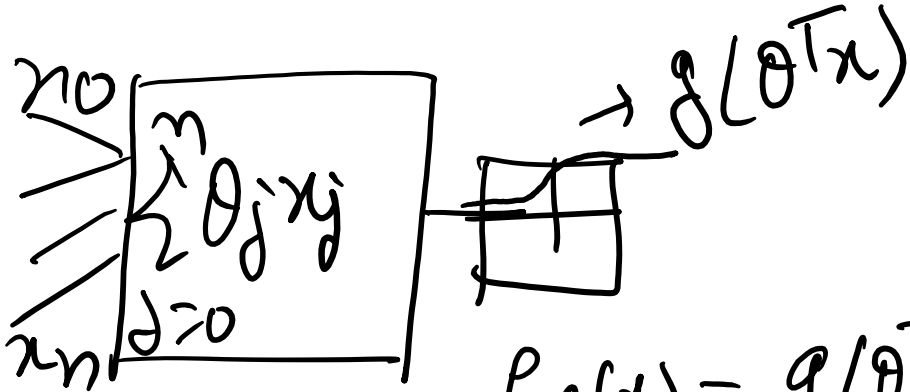
Neural Networks: Backpropagation

Apr 1, 2020

① Neural Networks :-

Last Class :- Perceptron





$h_\theta(x) = g(\theta^T x)$

↳ AND, OR, NAND

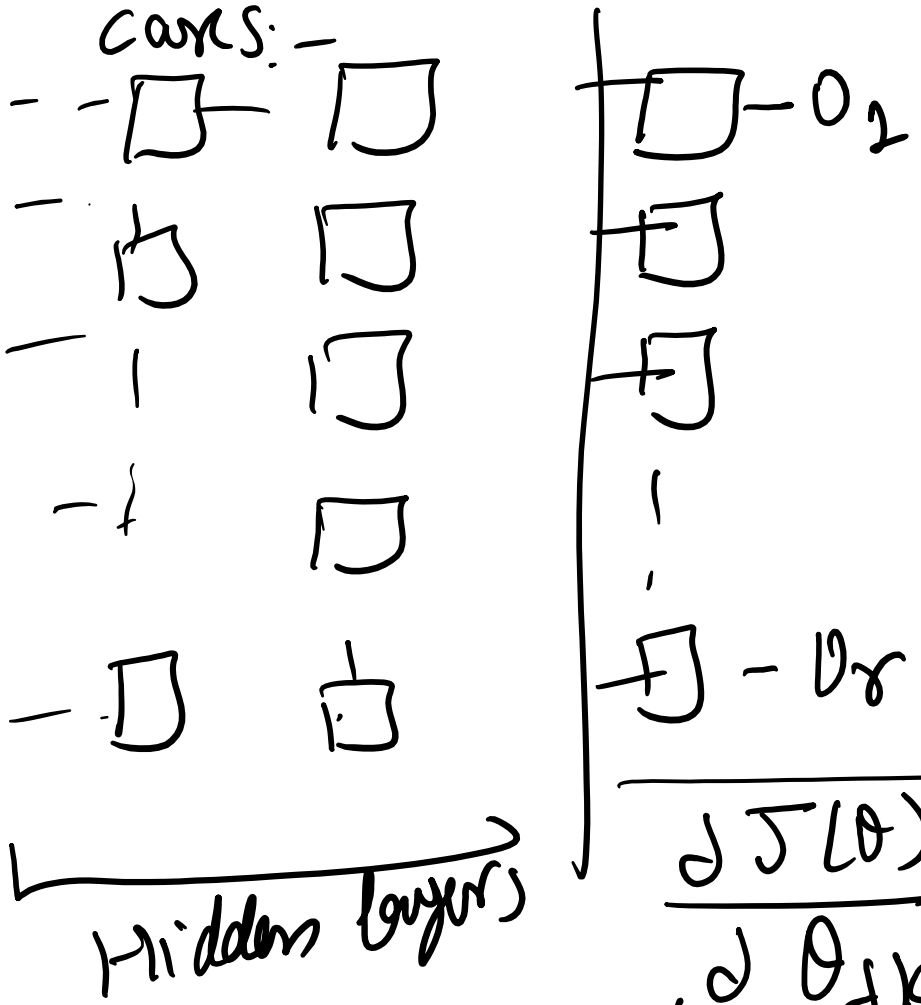
↳ XOR → Multiple such Perceptrons (arranged in a layered manner)

$$J(\theta) = \sum_{i=1}^{m} \sum_{\ell=1} \frac{(y_\ell^{(i)} - O_\ell)^2}{2m}$$

$$\frac{\nabla J(\theta)}{\partial \theta_{jk}}$$

θ_{jk} denotes weight of an interconnection

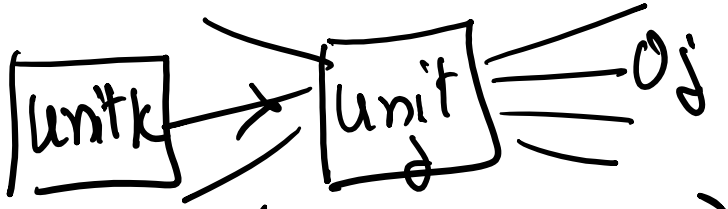
To compute the gradient, we will divide this problem into two cases:-



$$J(\theta) = \frac{1}{2} \sum_{e=1}^{m=1} (y_e - \theta_e)^2$$

$$O_j = g(net_j)$$

↳ sigmoid



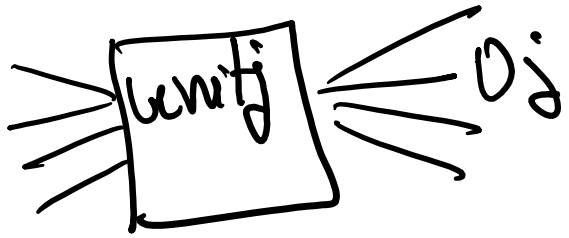
$$\frac{\partial J(\theta)}{\partial \theta_{jk}} \quad \forall jk$$

↳ $\frac{\partial J(\theta)}{\partial net_j}$

$$net_j = \sum_{k \in upNbrs(j)} \theta_{jk} x_{jk}$$

θ_{jk}:- weight of interconnection from unit k to j

x_{jk}:- input going from unit k to j



$$net_j = \sum_{k \in upNbrs(j)} \theta_{jk} x_{jk}$$

$$o_j = g(net_j)$$

Backpropagation

$$\frac{\partial J(\theta)}{\partial \theta_{jk}} = \frac{\partial J(\theta)}{\partial net_j} \cdot \left[\frac{\partial net_j}{\partial \theta_{jk}}\right] = \frac{\partial J(\theta)}{\partial net_j} \cdot x_{jk}$$

$$\hookrightarrow \frac{\partial \sum_{k \in upNbrs(j)} \theta_{jk} x_{jk}}{\partial \theta_{jk}} = x_{jk}$$

Our objective:-

$\frac{\partial J(\theta)}{\partial net_j}$ $\forall$ units j in the network

$$\frac{\partial o_j}{\partial net_j} = \frac{\partial g(net_j)}{\partial net_j} = g(net_j)(1 - g(net_j)) = o_j(1 - o_j)$$

(A) $j \in$ Output layer

(B) $j \in$ Hidden layer

(A) $\frac{\partial J(\theta)}{\partial net_j} = \frac{\partial \left[\frac{1}{2}\sum_{c=1}^{2}(y_c - o_c)^2\right]}{\partial net_j} = \frac{1}{2} * 2 * (y_j - o_j)(-1) * \frac{\partial o_j}{\partial net_j}$

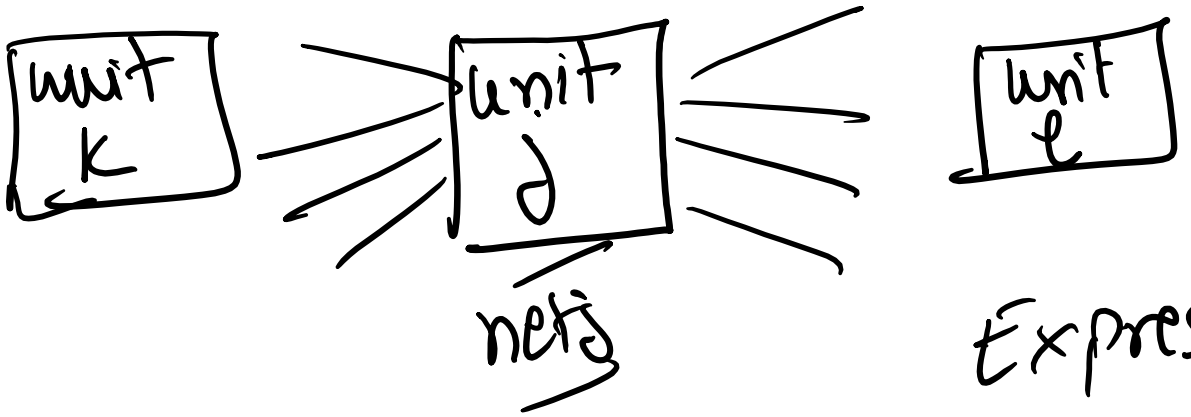
$= -(y_j - o_j) o_j (1 - o_j)$

↓ ε output layer

Define:- $\boxed{\delta_j = -\frac{\partial J(\theta)}{\partial net_j}}$

For output layers:- $\boxed{\delta_j = (y_j - o_j)\, o_j (1 - o_j)}$

(B) $\frac{\partial J(\theta)}{\partial net_j} \rightarrow \partial \varepsilon$ hidden layer



$J(\theta)$

Express $\frac{\partial J(\theta)}{\partial net_j}$ in terms of $\frac{\partial J(\theta)}{\partial net_l}$ where $l \in downNbr(j)$ → has already been computed

Aside:-

$f(y_1 \cdots y_k)$

↳ y_k is a function $x_1 \cdots x_n$

$f(y_1(x_1 \cdots x_n), y_2(x_1 \cdots x_n) \cdots y_k(x_1 \cdots x_n))$

Objective :- compute $\frac{\partial f}{\partial x_j}$ $\forall x_j$

$$\frac{\partial f(y_1 \cdots y_k)}{\partial x_j} = \sum_{k=1}^{K} \frac{\partial f(y_1 \cdots y_k)}{\partial y_k} \cdot \frac{\partial y_k}{\partial x_j}$$

Each y_k is a fn. of $(x_1 \cdots x_n)$

extension when dependence is through multiple variables

$$\frac{f(y)}{\partial x} = \frac{\partial f(y)}{\partial y} \cdot \frac{\partial y}{\partial x}$$ → standard chain rule

y is a fn of x

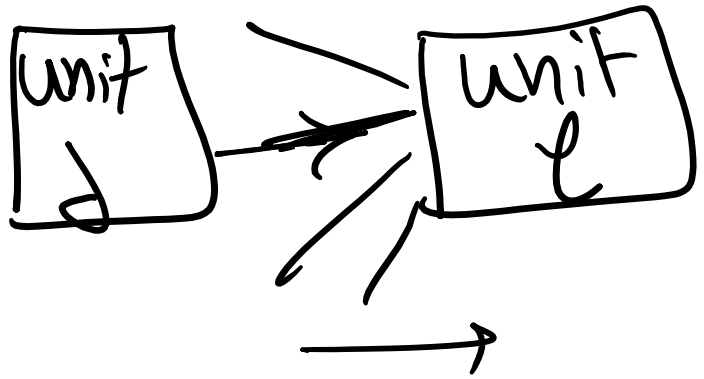
→ Back to computing

$$\rightarrow \frac{\partial J(\theta)}{\partial net_j} = \sum_{e \in \text{downNbrs}(j)} \left[\frac{\partial J(\theta)}{\partial net_e} \cdot \right] \left[\frac{\partial net_e}{\partial net_j}\right]$$

already computed

$$\frac{\partial net_e}{\partial net_j} = \frac{\partial \sum_{j \in upNbr(e)} x_{ej}\,\theta_{ej}}{\partial net_j}$$

$x_{ej} \equiv o_j$



$$o_j = g(net_j)$$

$$o_j \equiv x_{ej}$$

$$= \frac{\partial \sum_{j \in upNbr(e)} o_j \cdot \theta_{ej}}{\partial net_j} = \frac{\partial\, o_j \cdot \theta_{ej}}{\partial net_j} = \theta_{ej} \cdot \frac{\partial o_j}{\partial net_j}$$

$$\frac{\partial net_e}{\partial net_j} = \theta_{ej} \cdot (o_j)(1 - o_j)$$

(B) Hidden layer

$$\frac{\partial J(\theta)}{\partial net_j} = \sum_{e \in downNbr(j)} \left[\frac{\partial J(\theta)}{\partial net_e}\right] \cdot \left[\frac{\partial net_e}{\partial net_j}\right] \qquad \theta_{ej}\, o_j(1-o_j)$$

$$\equiv \sum_{e \in downNbr(j)} -\delta_e \cdot \theta_{ej}\, o_j(1-o_j)$$

(A) $$\frac{\partial J(\theta)}{\partial net_j} = (y_j - o_j)(-1)\, o_j(1-o_j)$$

$$\delta_e = -\frac{\partial J(\theta)}{\partial net_e}$$

$$\frac{\partial J(\theta)}{\partial net_j} \rightarrow (A)\ (y_j - o_j)(-1)\, o_j(1-o_j)$$

$$\rightarrow (B) \sum_{e \in downNbr(j)} -\delta_e\, \theta_{ej}\, o_j(1-o_j)$$

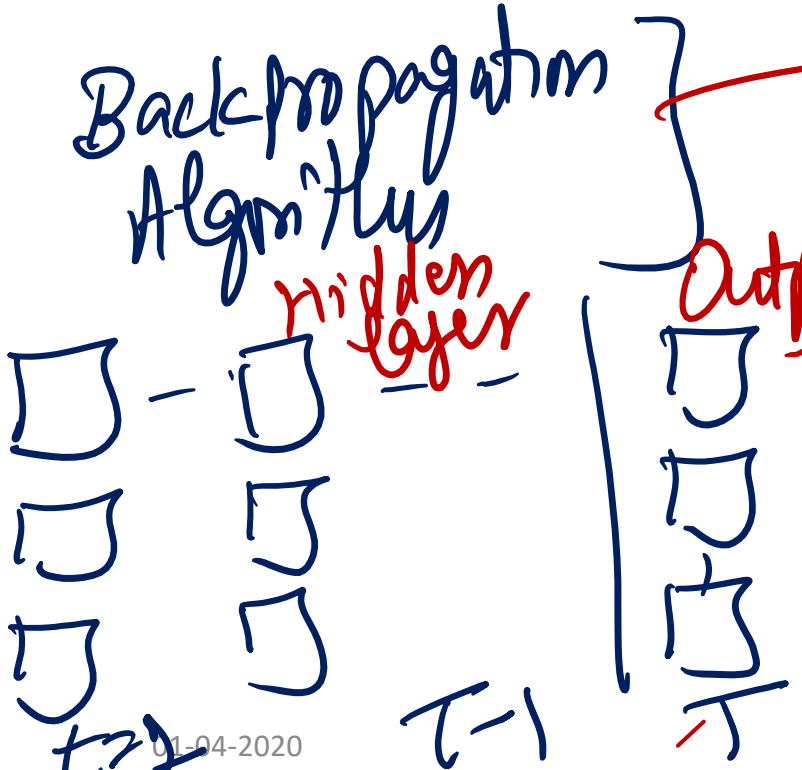
$$\delta_j = -\frac{\partial J(\theta)}{\partial net_j}$$

$\frac{\partial J(\theta)}{\partial \theta_{jk}}$ — As described earlier

(A) output layer: $\delta_j = (y_j - o_j)\, o_j (1 - o_j)$

(B) Hidden layers: $\delta_j = \sum_{e \in downNbr(j)} \delta_e \theta_{ej}\, o_j (1 - o_j)$

Backpropagation Algorithm



(A) Output layer: Compute δ_j $\forall j \in layer(T)$

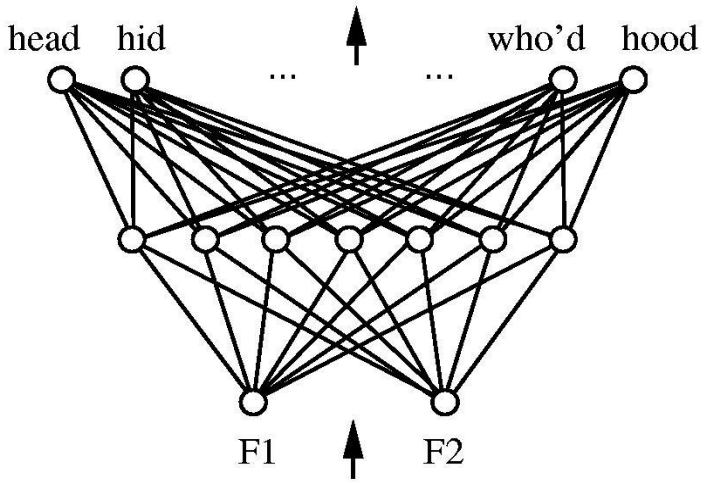
(B) For $t = T-1$ to 1

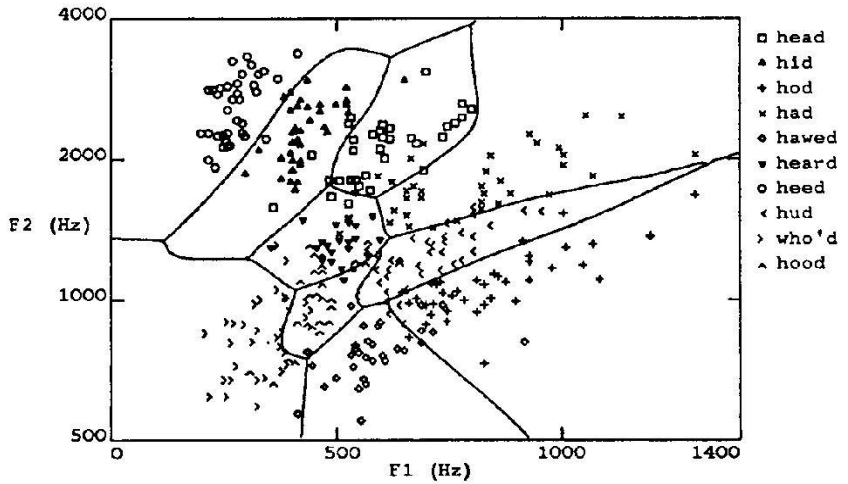
Hidden layers: compute δ_j $\forall j \in layer(t)$ in terms of δ_j' $\forall j' \in layer(t+1)$

3

NOTE: Following slides have been borrowed from Pedro Domingos' Machine Learning course offered at the University of Washington

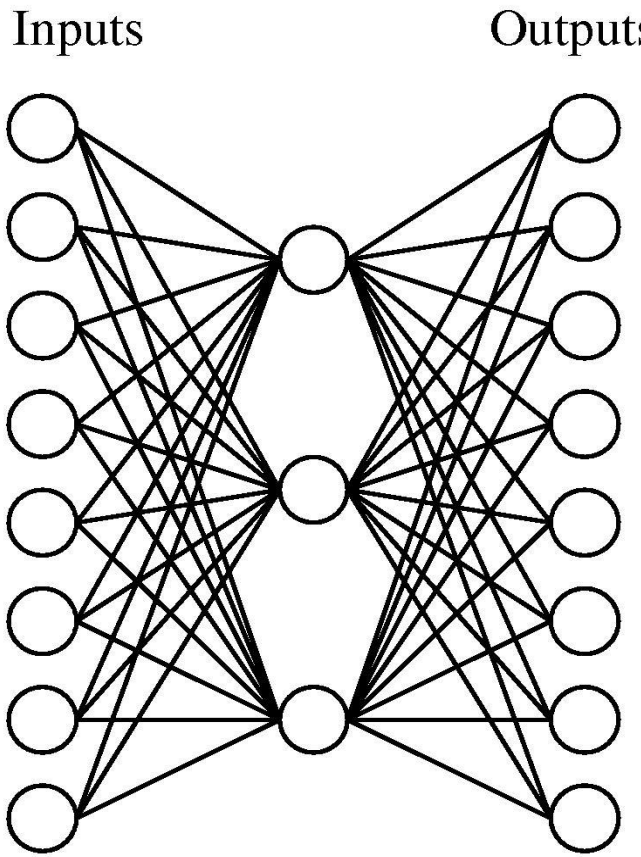
Multilayer Networks of Sigmoid Units





Source: Pedro Domingos' course offered at Univ. of Washington

Learning Hidden Layer Representations



Source: Pedro Domingos' course offered at Univ. of Washington

A target function:

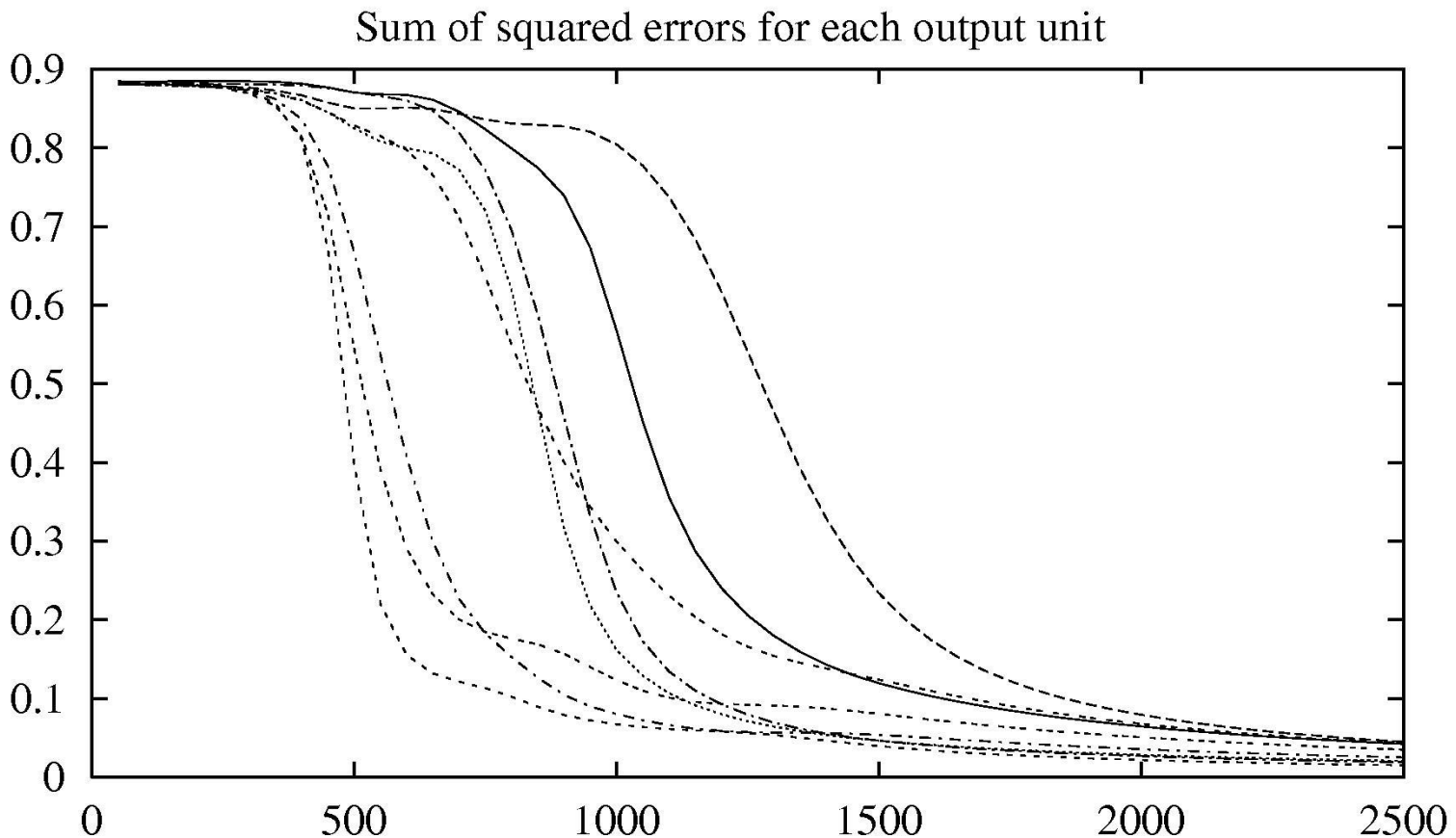
Input		Output
10000000	$\rightarrow$	10000000
01000000	$\rightarrow$	01000000
00100000	$\rightarrow$	00100000
00010000	$\rightarrow$	00010000
00001000	$\rightarrow$	00001000
00000100	$\rightarrow$	00000100
00000010	$\rightarrow$	00000010
00000001	$\rightarrow$	00000001

Can this be learned?

Learned hidden layer representation:

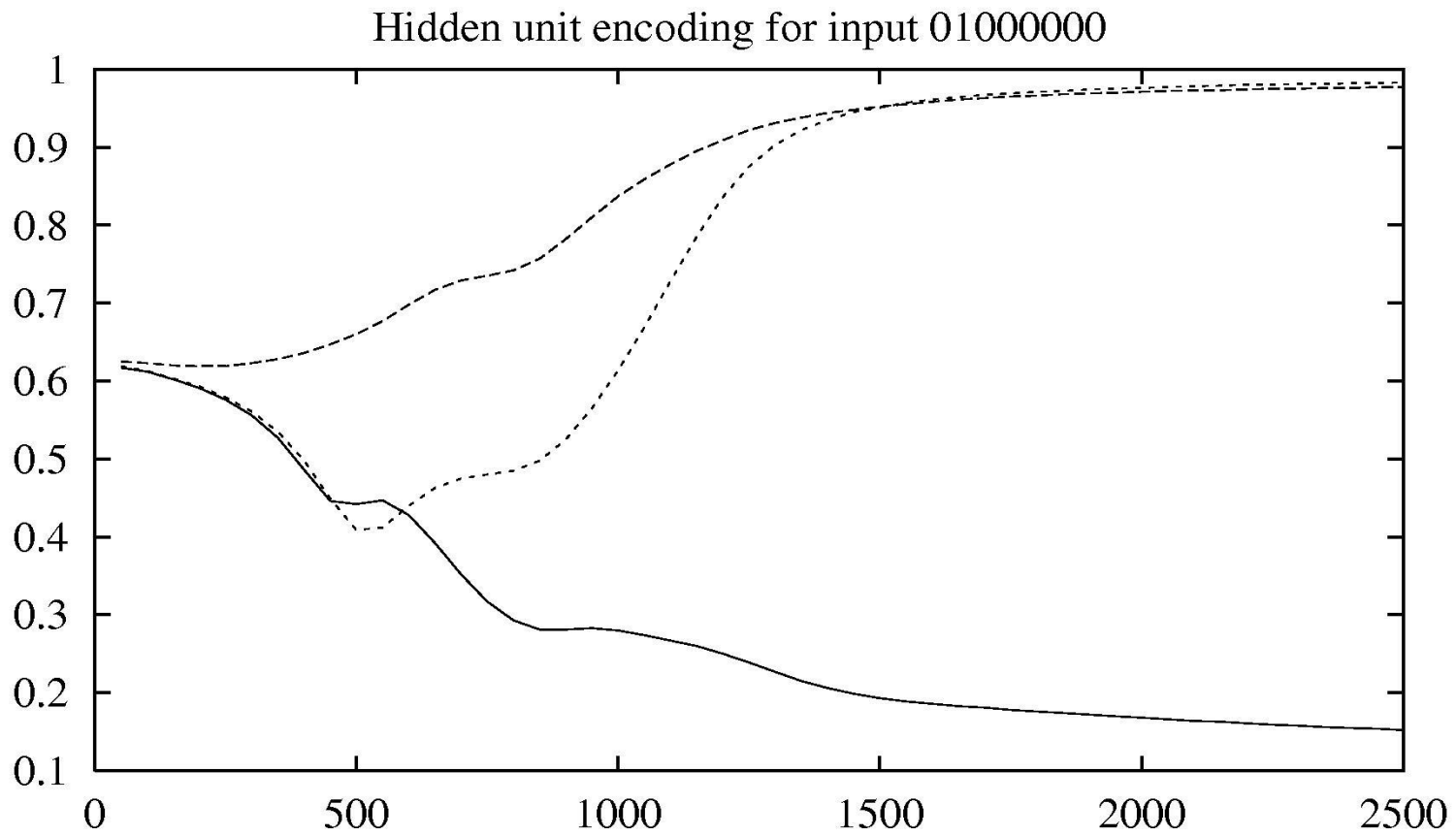
Input		Hidden Values				Output
10000000	→	.89	.04	.08	→	10000000
01000000	→	.01	.11	.88	→	01000000
00100000	→	.01	.97	.27	→	00100000
00010000	→	.99	.97	.71	→	00010000
00001000	→	.03	.05	.02	→	00001000
00000100	→	.22	.99	.99	→	00000100
00000010	→	.80	.01	.98	→	00000010
00000001	→	.60	.94	.01	→	00000001

Training



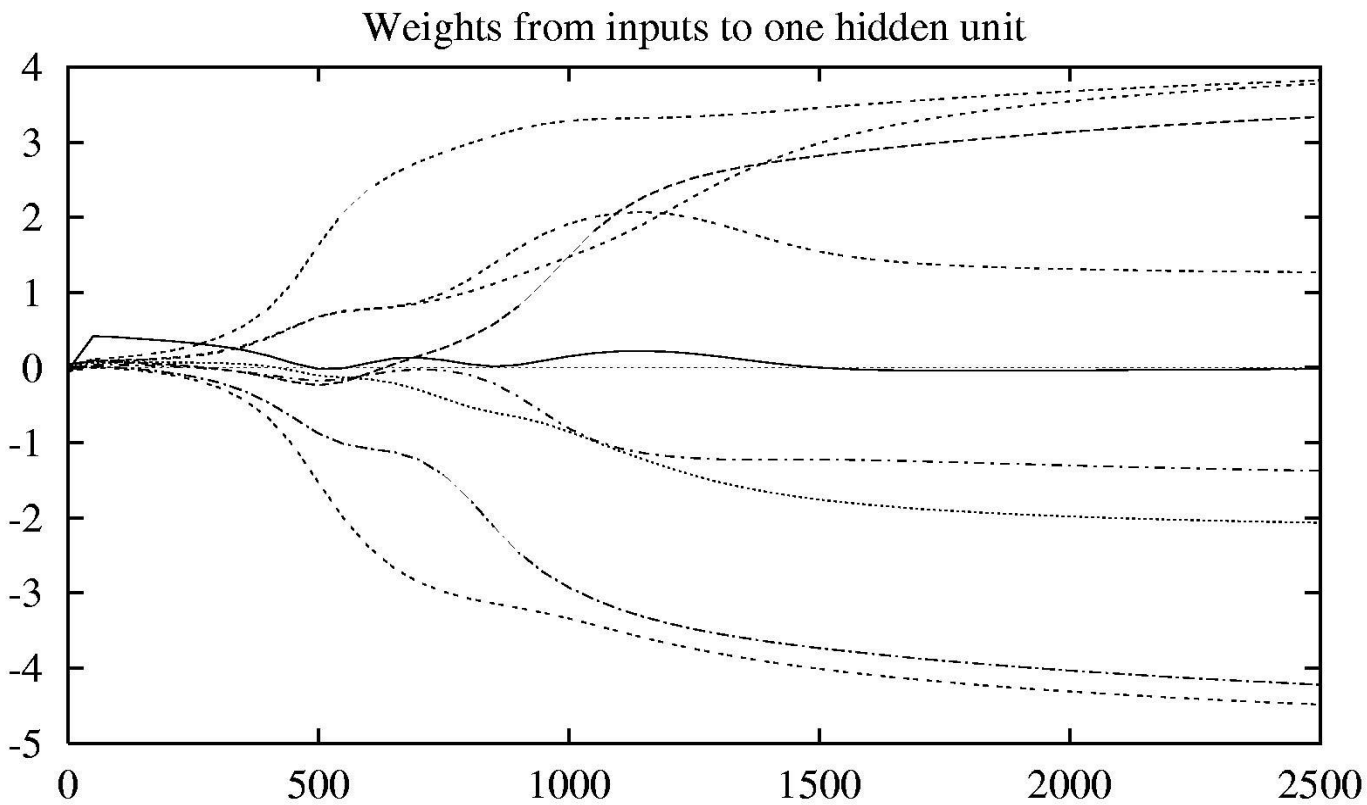
Source: Pedro Domingos' course offered at Univ. of Washington

Training



Source: Pedro Domingos' course offered at Univ. of Washington

Training



Source: Pedro Domingos' course offered at Univ. of Washington

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

Source: Pedro Domingos' course offered at Univ. of Washington

Expressiveness of Neural Nets

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
- Any function can be approximated to arbitrary accuracy by a network with two hidden layers