Stochastic (Incremental) Gradient Descent

Batch gradient descent (GD):
Do until satisfied

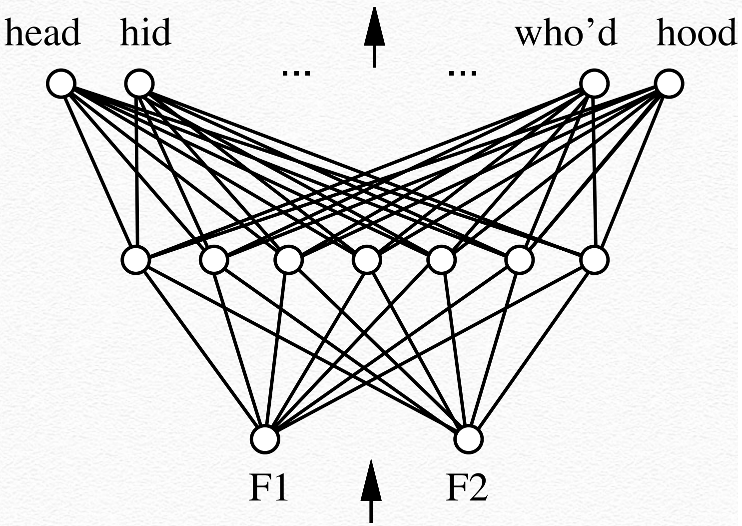
1. Compute gradient $\nabla L_D(\mathbf{w})$
2. $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla L_D(\mathbf{w})$ where
   $L_D(\mathbf{w}) = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$

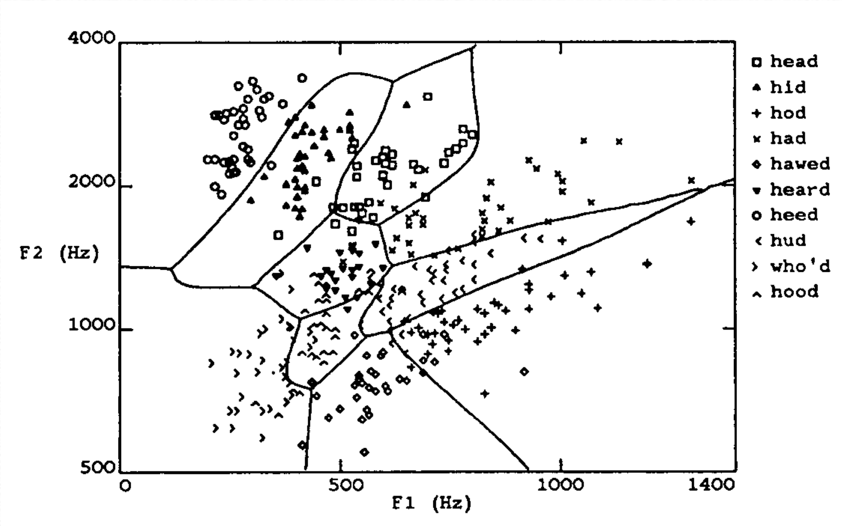
Stochastic gradient descent (SGD):
Do until satisfied

- For each training example $d \in D$
  1. Compute gradient $\nabla L_d(\mathbf{w})$
  2. $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla L_d(\mathbf{w})$ where
     $L_d(\mathbf{w}) = \frac{1}{2} (t_d - o_d)^2$

- SGD can approximate batch GD arbitrarily closely if learning rate η is sufficiently small
- General case. Objective function (differentiable wrt model parameters $\mathbf{w}$) can be decomposed into a sum of terms, each depending on a subset of training examples

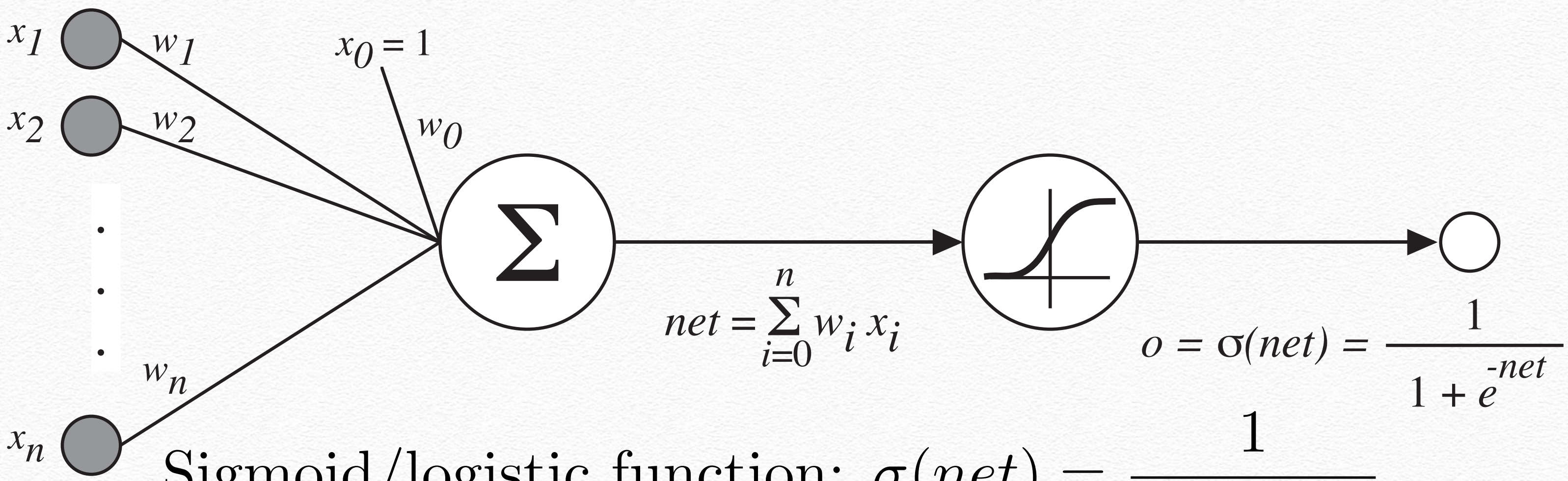
Multilayer Networks of Sigmoid Units





Speech recognition task.

Sigmoid Unit



Sigmoid/logistic function: $\sigma(net) = \dfrac{1}{1 + e^{-net}}$

Useful property: $\dfrac{\mathrm{d}\sigma(net)}{\mathrm{d}net} = \sigma(net)(1 - \sigma(net))$

GD rules can be derived to train

- 1 sigmoid unit
- Multilayer networks of sigmoid units → Backpropagation

Error/Loss Gradient for 1 Sigmoid Unit

$$
\begin{aligned}
\frac{\partial L_D}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 \\
&= \frac{1}{2} \sum_d \frac{\partial}{\partial w_i} (t_d - o_d)^2 \\
&= \frac{1}{2} \sum_d 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d) \\
&= \sum_d (t_d - o_d) \left(-\frac{\partial o_d}{\partial w_i} \right) \\
&= -\sum_d (t_d - o_d) \frac{\partial o_d}{\partial net_d} \frac{\partial net_d}{\partial w_i} \\
\frac{\partial o_d}{\partial net_d} &= \frac{\partial \sigma(net_d)}{\partial net_d} = o_d(1 - o_d) \\
\frac{\partial net_d}{\partial w_i} &= \frac{\partial (\mathbf{w} \cdot \mathbf{x}_d)}{\partial w_i} = x_{id} \\
\frac{\partial L_D}{\partial w_i} &= -\sum_{d \in D} (t_d - o_d) o_d (1 - o_d) x_{id}
\end{aligned}
$$

Feedforward Networks of Sigmoid Units

Use GD to learn **w** that minimizes squared error/loss:

$$L_D(\mathbf{w}) = \frac{1}{2}\sum_{d \in D}\sum_{k \in K}(t_{kd} - o_{kd})^2$$

where K is the set of output units in the network, t_{kd} and o_{kd} are, respectively, target output and output of sigmoid unit associated with k-th output unit and training example d

Backpropagation algorithm here assumes 2 layers of sigmoid units and is based on SGD:

$$L_d(\mathbf{w}) = \frac{1}{2}\sum_{k \in K}(t_k - o_k)^2$$

BACKPROPAGATION Algorithm

Idea. Initialize $\mathbf{w}$ randomly, propagate input forward and then errors backward thru the network for each training example

Initialize all network weights to small random numbers
Until satisfied, do

- For each training example $\langle \mathbf{x}, (t_k)^{\top}_{k \in K} \rangle$, do
  1. Input instance $\mathbf{x}$ to the network and compute output of every sigmoid unit in the hidden and output layers
  2. For each output unit k, compute error $\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$
  3. For each hidden unit h, compute error $\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in K} w_{hk} \delta_k$
  4. Update each weight $w_{hk} \leftarrow w_{hk} + \Delta w_{hk}$ where $\Delta w_{hk} = \eta \delta_k o_h$
  5. Update each weight $w_{ih} \leftarrow w_{ih} + \Delta w_{ih}$ where $\Delta w_{ih} = \eta \delta_h x_i$

Derivation of BACKPROPAGATION

$$
\begin{aligned}
\frac{\partial L_d}{\partial w_{hk}} &= \frac{\partial L_d}{\partial o_k}\frac{\partial o_k}{\partial net_k}\frac{\partial net_k}{\partial w_{hk}} \text{ where } net_k = \sum_{h'} w_{h'k} o_{h'} \\
\frac{\partial L_d}{\partial o_k} &= \frac{\partial}{\partial o_k}\frac{1}{2}\sum_{k' \in K}(t_{k'} - o_{k'})^2 = -(t_k - o_k) \\
\frac{\partial o_k}{\partial net_k} &= \frac{\partial \sigma(net_k)}{\partial net_k} = o_k(1 - o_k) \\
\frac{\partial net_k}{\partial w_{hk}} &= o_h \\
\Delta w_{hk} &= -\eta \frac{\partial L_d}{\partial w_{hk}} = \eta(t_k - o_k)o_k(1 - o_k)o_h
\end{aligned}
$$

Homework. Derive $\Delta w_{ih} = -\eta \partial L_d / \partial w_{ih} = \eta \delta_h x_i$.

Remarks on BACKPROPAGATION

- L_D has multiple local minima! GD is guaranteed to converge to some local min., but not necessarily global min.
  - In practice, GD often performs well, especially after using multiple random initializations of **w**
- Often include weight momentum $\alpha \in [0,1)$:

$$\Delta w_{hk} \leftarrow \eta \delta_k o_h + \alpha \Delta w_{hk} \ , \quad \Delta w_{ih} \leftarrow \eta \delta_h x_i + \alpha \Delta w_{ih}$$

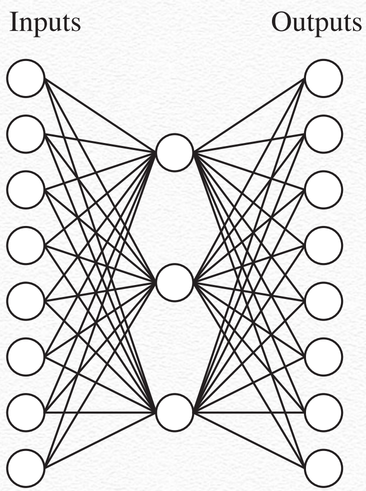
- Easily generalized to feedforward networks of arbitrary depth:
  - Step 3: Let K denote all units in the next deeper layer whose inputs include output of h
  - Step 5: Let x_i denote the output of unit i in previous layer that is input to h

Remarks on BACKPROPAGATION

- Expressive hypothesis space. Requires limited depth feedforward networks:
  - Every Boolean function can be represented by a network with one hidden layer but may require exponential hidden units in no. of inputs
  - Every bounded continuous function can be approximated with arbitrarily small error by a 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)
- Approximate inductive bias. Smooth interpolation between data points.

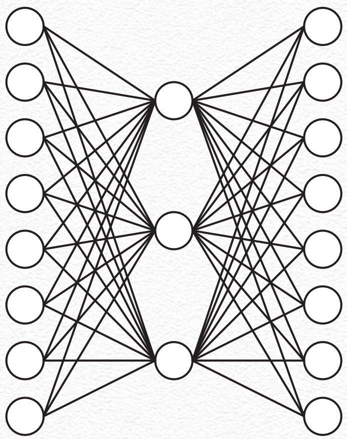
Learning Hidden Layer Representations

Can the $8 \times 3 \times 8$ feedforward network (left) be trained using BACKPROPAGATION to learn the target function (right)?



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

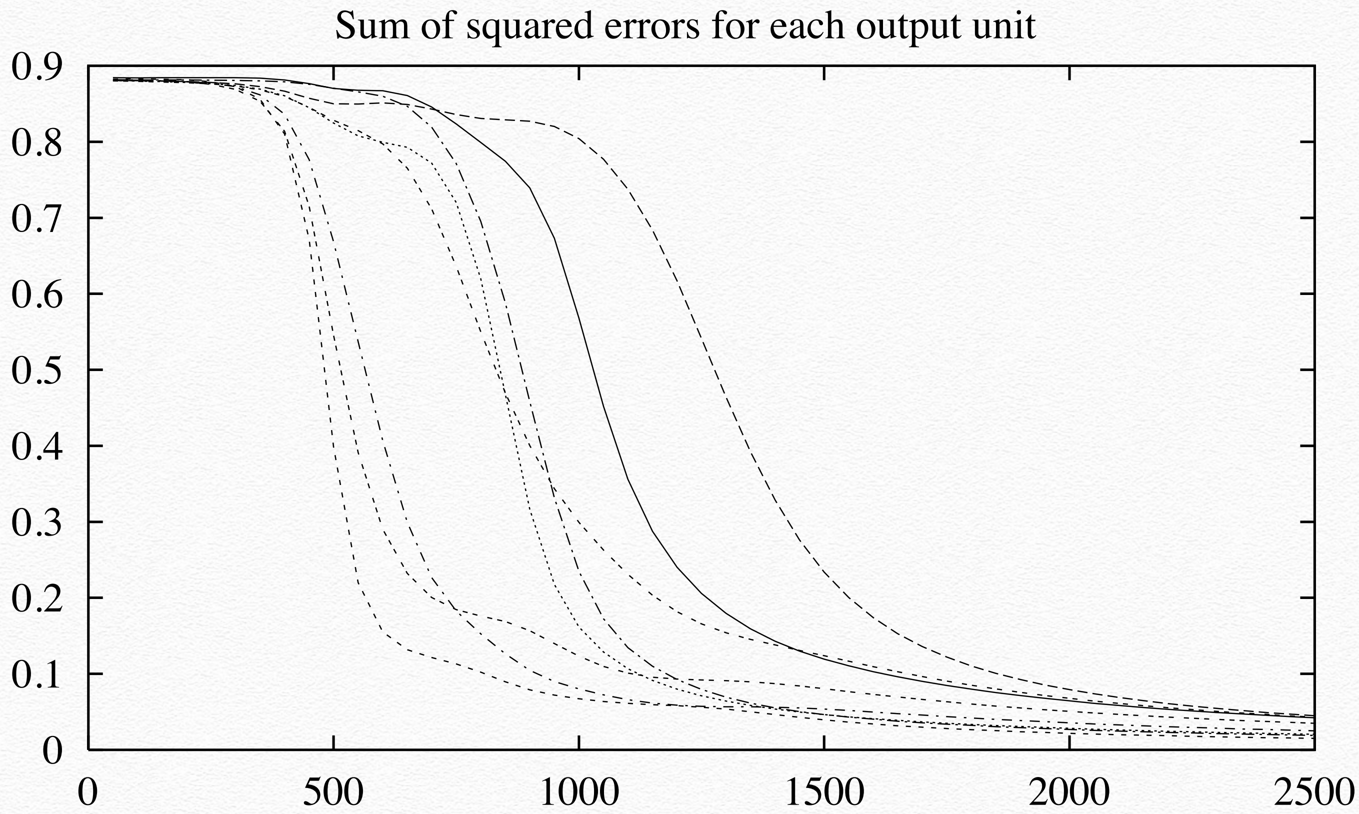
Learning Hidden Layer Representations



Learned hidden layer representation

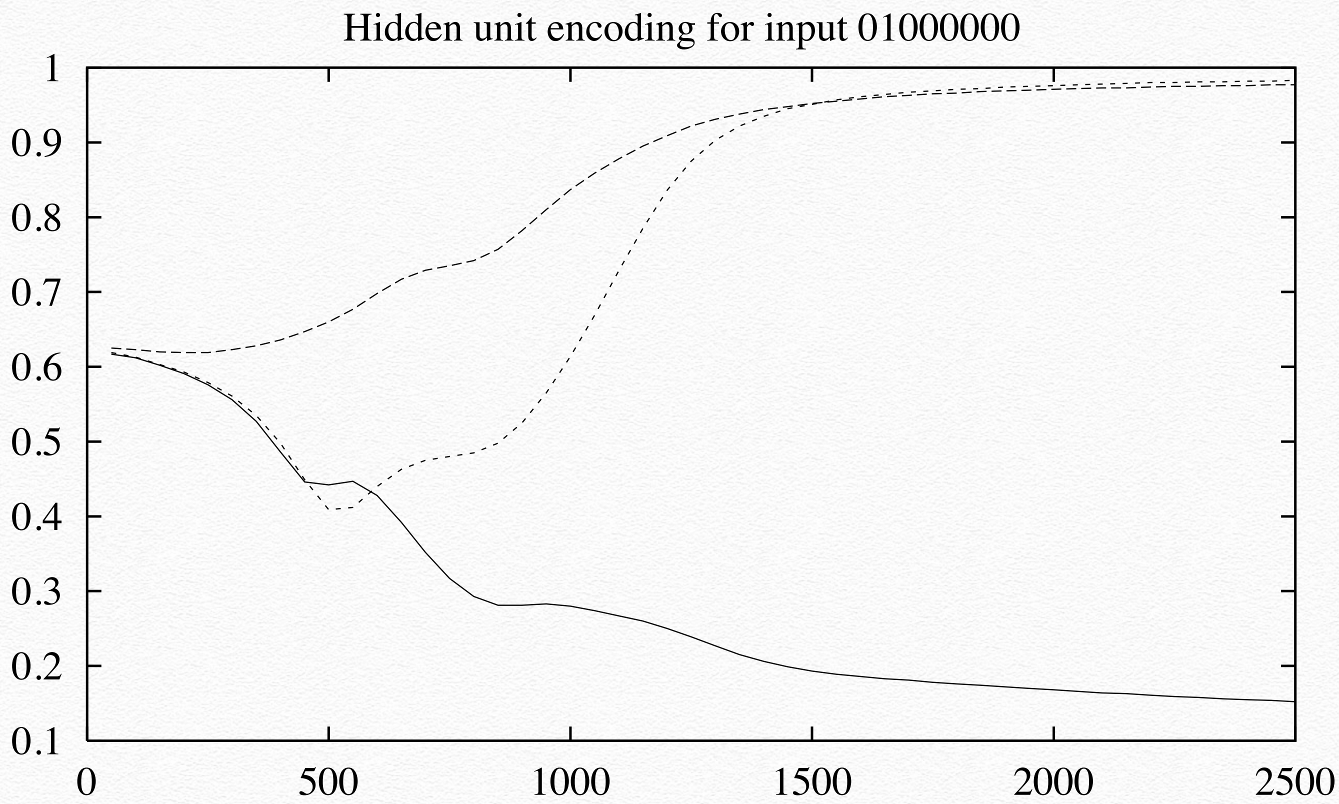
Input		Hidden Values				Output
10000000	$\rightarrow$	.89	.04	.08	$\rightarrow$	10000000
01000000	$\rightarrow$	.15	.99	.99	$\rightarrow$	01000000
00100000	$\rightarrow$	.01	.97	.27	$\rightarrow$	00100000
00010000	$\rightarrow$	.99	.97	.71	$\rightarrow$	00010000
00001000	$\rightarrow$	.03	.05	.02	$\rightarrow$	00001000
00000100	$\rightarrow$	.01	.11	.88	$\rightarrow$	00000100
00000010	$\rightarrow$	.80	.01	.98	$\rightarrow$	00000010
00000001	$\rightarrow$	.60	.94	.01	$\rightarrow$	00000001

Training

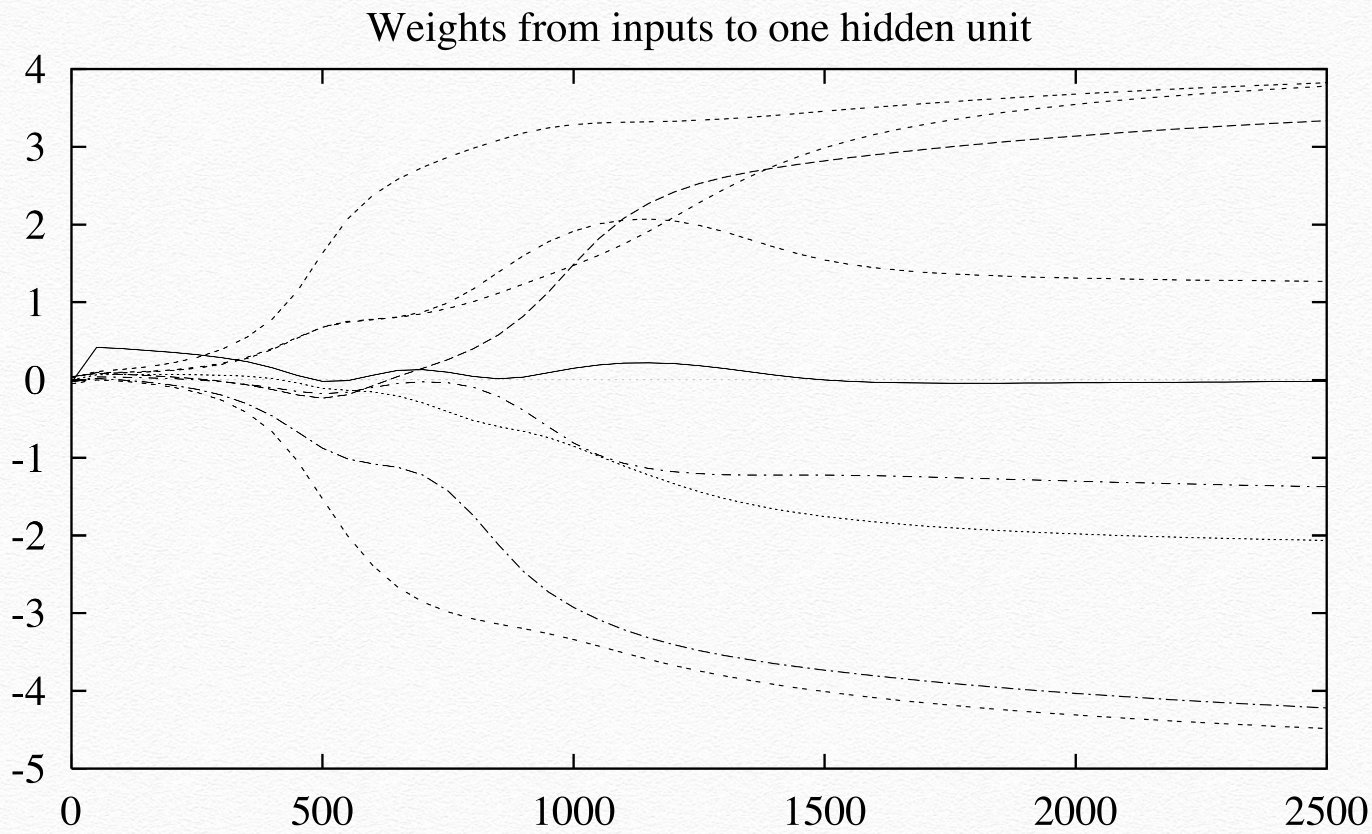


Sum of squared errors for each output unit
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
500
1000
1500
2000
2500

Training

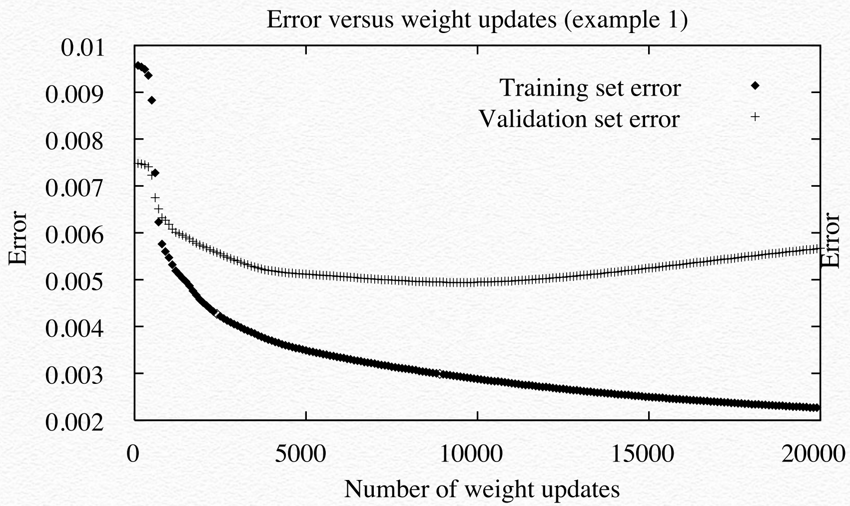

Hidden unit encoding for input 01000000
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
500
1000
1500
2000
2500

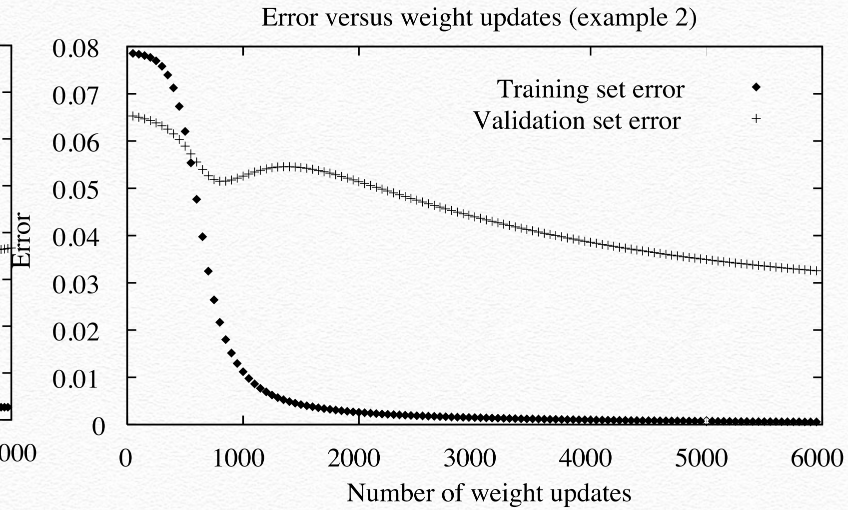
Training



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

Overfitting





Alternative Loss/Error Functions

- Penalize large weights:

$$L_D(\mathbf{w}) = \frac{1}{2} \sum_{d \in D} \sum_{k \in K} (t_{kd} - o_{kd})^2 + \gamma \sum_{j,\ell} w_{j\ell}^2$$

- Train on target values as well as slopes:

$$L_D(\mathbf{w}) = \frac{1}{2} \sum_{d \in D} \sum_{k \in K} \left[\boxed{(t_{kd} - o_{kd})^2} + \mu \boxed{\sum_{i=1}^{n} \left(\frac{\partial t_{kd}}{\partial x_{id}} - \frac{\partial o_{kd}}{\partial x_{id}} \right)^2} \right]$$

- Tie together weights (e.g., phoneme recognition networks)