

# Machine Learning 10-701

Tom M. Mitchell  
Machine Learning Department  
Carnegie Mellon University

March 24, 2011

## Today:

- Non-linear regression
- Artificial neural networks
- Backpropagation
- Cognitive modeling
- Deep belief networks

## Reading:

- Mitchell: Chapter 4
- Bishop: Chapter 5

## 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 logistic units
- Each unit is a logistic function

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

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

# Multilayer Networks of Sigmoid Units

---

The diagram illustrates a multilayer network of sigmoid units (neurons) and a corresponding scatter plot of frequency components (F1 and F2) for various words.

**Network Diagram:**

- The network consists of three layers: an input layer (F1), a hidden layer (F2), and an output layer.
- Input nodes are labeled: head, hid, ..., who'd, hood.
- Hidden nodes are labeled: F1, F2.
- Output nodes are labeled: head, hid, ..., who'd, hood.
- Connections are shown between nodes in adjacent layers. Red lines highlight specific connections, particularly those involving the 'who'd' and 'hood' nodes.

**Scatter Plot:**

- The plot shows F1 (Hz) on the x-axis (0 to 1400) and F2 (Hz) on the y-axis (500 to 4000).
- Data points are categorized by word, represented by different symbols: head (circle), hid (triangle), hod (plus), had (x), haved (asterisk), heard (diamond), heed (square), hud (asterisk), who'd (less-than), hood (greater-than).
- A red outline highlights a cluster of points corresponding to the words 'who'd' and 'hood'.



## 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

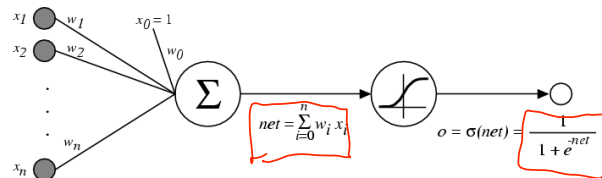
→ much parallel computation

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 → Backpropagation

## M(C)LE Training for Neural Networks

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

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

- 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  $Y$

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

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

deterministic

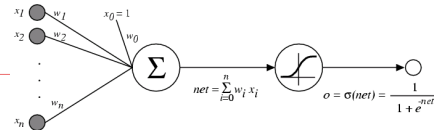
Gaussian  $P(W) = N(0, \sigma I)$

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

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

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

## Error Gradient for a Sigmoid Unit



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

But we know:

$$\begin{aligned}\frac{\partial o_d}{\partial net_d} &= \frac{\partial \sigma(net_d)}{\partial net_d} = o_d(1 - o_d) \\ \frac{\partial net_d}{\partial w_i} &= \frac{\partial (\vec{w} \cdot \vec{x}_d)}{\partial w_i} = x_{i,d}\end{aligned}$$

So:

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

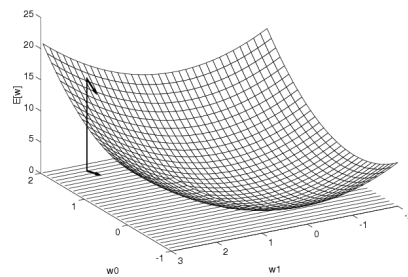
$x_d$  = input

$t_d$  = target output

$o_d$  = observed unit output

$w_i$  = weight i

## 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.,

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

## Incremental (Stochastic) Gradient Descent

### Batch mode Gradient Descent:

Do until satisfied

1. Compute the gradient  $\nabla E_D[\vec{w}]$
2.  $\vec{w} \leftarrow \vec{w} - \eta \nabla E_D[\vec{w}]$

*data = D*

### → 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 (MLE)

✓ 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  $h$

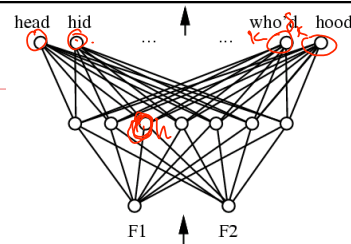
$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in \text{outputs}} w_{h,k} \delta_k$$

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

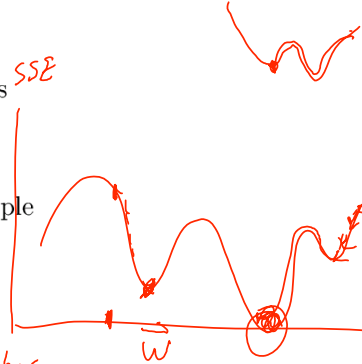
$o_d$  = observed unit output

$w_{ij}$  = wt from  $i$  to  $j$

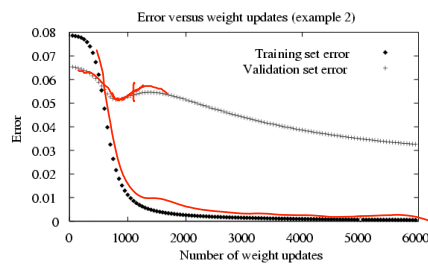
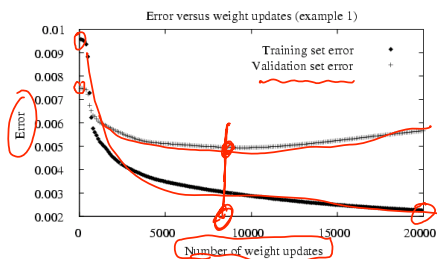
## More on Backpropagation

- 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*  $\alpha$ 

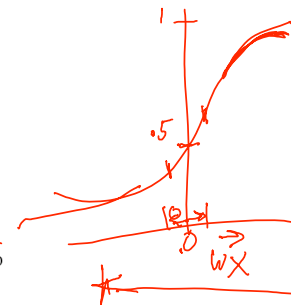
$$\Delta w_{i,j}(n) = \underbrace{\eta}_{\text{iter}} \delta_j x_{i,j} + \underbrace{\alpha}_{\text{prev iter update}} \Delta w_{i,j}(n-1)$$
- Minimizes error over *training* examples
  - Will it generalize well to subsequent examples?
- Training can take thousands of iterations → slow!
- Using network after training is very fast



## Overfitting in ANNs



$$\frac{1}{1 + \exp(-wx)}$$



## Dealing with Overfitting

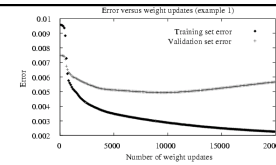
Our learning algorithm involves a parameter

$n$  = number of gradient descent iterations

How do we choose  $n$  to optimize future error?

(note: similar issue for logistic regression, decision trees, ...)

e.g. the  $n$  that minimizes error rate of neural net over future data



## Dealing with Overfitting

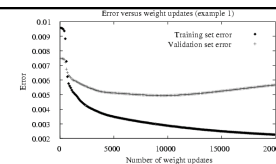
Our learning algorithm involves a parameter

$n$  = number of gradient descent iterations

How do we choose  $n$  to optimize future error?

- Separate available data into training and validation set
- Use training to perform gradient descent
- $n \leftarrow$  number of iterations that optimizes validation set error

→ gives *unbiased estimate of optimal  $n$*   
(but a biased estimate of true error)





## K-Fold Cross Validation

Idea: train multiple times, leaving out a disjoint subset of data each time for test. Average the test set accuracies.

---

Partition data into K disjoint subsets

For k=1 to K

    testData = kth subset

$h \leftarrow$  classifier trained\* on all data except for testData

    accuracy(k) = accuracy of h on testData

end

FinalAccuracy = mean of the K recorded testset accuracies

\* might withhold some of this to choose number of gradient decent steps

## Leave-One-Out Cross Validation

This is just k-fold cross validation leaving out one example each iteration

---

Partition data into K disjoint subsets, each containing one example

For k=1 to K

    testData = kth subset

$h \leftarrow$  classifier trained\* on all data except for testData

    accuracy(k) = accuracy of h on testData

end

FinalAccuracy = mean of the K recorded testset accuracies

\* might withhold some of this to choose number of gradient decent steps

## Expressive Capabilities of ANNs

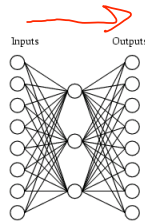
Boolean functions:

- 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

## Learning Hidden Layer Representations



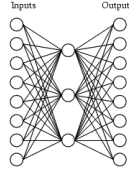
A target function:

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

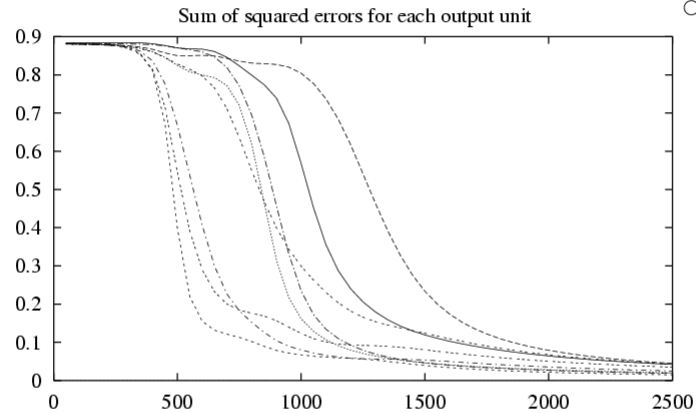
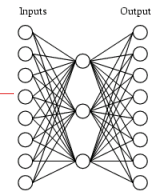
A network:



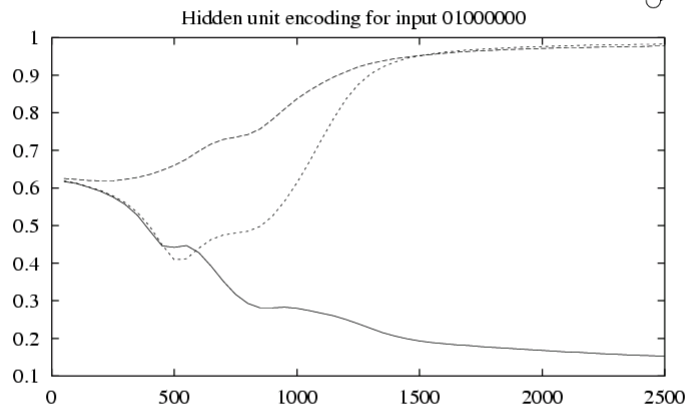
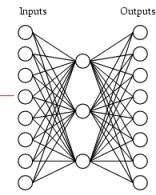
Learned hidden layer representation:

Input	Hidden Values	Output
10000000	→ <u>.89</u> .04 .08	→ 10000000
01000000	→ .01 <u>.11</u> .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

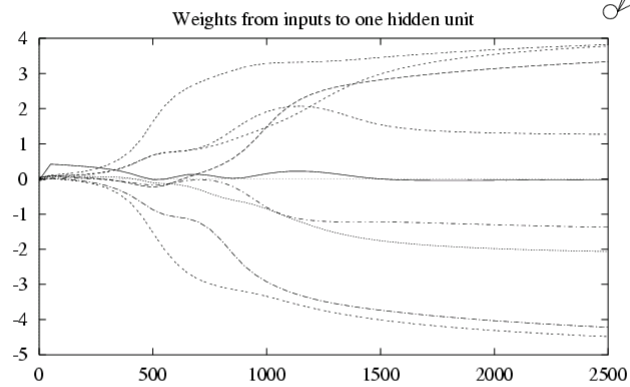
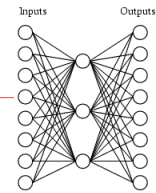
## Training



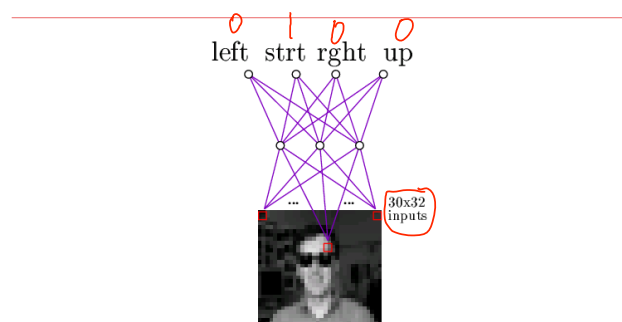
## Training



## Training



## Neural Nets for Face Recognition

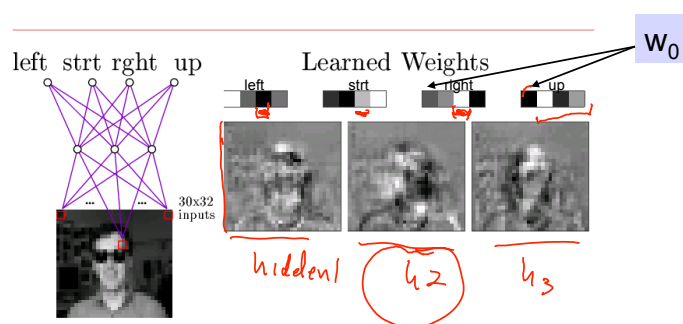


Typical input images



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>