Comp 135
Introduction to Machine Learning and Data Mining

Fall 2015

Professor: Roni Khardon

Computer Science
Tufts University

Recall Linear Threshold Units

- The basic model:

$$\text{Output} = f(\sum w_j x_j)$$

- Where f can be one of

$$f = \text{sign}() \text{ Value in } \{-1, 1\}$$

$$f = \text{step}() \text{ Value in } \{0, 1\}$$

$$\sigma(a) = \frac{1}{1+e^{-a}}$$

$$p(f = 1) = \sigma(\sum w_j x_j)$$

- Note: we are focusing on one example and omitting the index i saying that this is the i'th example to avoid clutter in notation

Linear Sigmoid Units

- Today we will work with units whose output is a real value in [0,1]

$$\text{Output} = \hat{y} = \sigma(\sum w_j x_j)$$

$$\sigma(a) = \frac{1}{1+e^{-a}}$$

- This conveniently satisfies

$$\sigma'(a) = \frac{-(-e^{-a})}{(1+e^{-a})^2} = \sigma(a)(1 - \sigma(a))$$

Linear Sigmoid Units

- Consider an example (x,y)
- And error function

$$\text{Err} = \frac{1}{2}[y - \hat{y}]^2 = \frac{1}{2}[y - \sigma(\sum_j w_j x_j)]^2$$

- Applying gradient descent

$$w_k = w_k - \eta \frac{\partial \text{Err}}{\partial w_k}$$

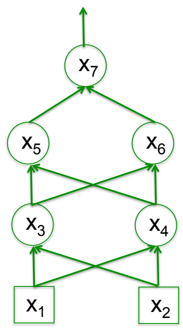
- We get the update rule

$$w_k = w_k + \eta(y - \hat{y})\hat{y}(1 - \hat{y})x_k$$

Multi Layer Networks

- Must first develop convenient notation
- This is different from single unit notation
- But it simplifies the exposition of the algorithm that follows

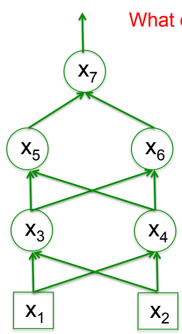
Multi Layer Networks



Multi Layer Networks

- Must first develop convenient notation
- Denote input as before by $x_1, \ldots, x_n$
- An internal node is identified by its index i, and its output is x_i
- All internal nodes are $x_{n+1}, \ldots, x_N$
- And the final output is x_N
- The link from unit j to i has weight $w_{j,i}$
- The sum at unit i is $s_i = \sum_j w_{j,i} x_j$
- The output at i is $x_i = \sigma(s_i) = \sigma(\sum w_{j,i} x_j)$

Multi Layer Networks



What does the network predict on this example?

$$\eta = 0.1$$
$$w_{56} = w_{67} = 1$$
$$w_{35} = w_{36} = w_{45} = w_{46} = 0.6$$
$$w_{13} = w_{14} = w_{23} = w_{24} = 1$$

Input example : $(x_1, x_2) = (2, 3)$

Desired output : $L = 0$

Example

First Step: compute $s_i, x_i,$ and $\sigma_i' = x_i(1 - x_i)$

$s_3 = 1 * 2 + 1 * 3 = 5$	$x_3 = \frac{1}{1+e^{-5}} = 0.993$	$\sigma_3' = 0.007$
$s_4 = 5$	$x_4 = 0.993$	$\sigma_4' = 0.007$
$s_5 = 0.6 * 0.993 +$ $0.6 * 0.993 = 1.192$	$x_5 = 0.767$	$\sigma_5' = 0.179$
$s_6 = 1.192$	$x_6 = 0.767$	$\sigma_6' = 0.179$
$s_7 = 1 * 0.767 +$ $1 * 0.767 = 1.534$	$x_7 = \frac{1}{1+e^{-1.534}} = 0.823$	$\sigma_6' = 0.146$

Multi Layer Networks

- As before we get an example (x,y).
- x specifies the input units $x_1, \ldots, x_n$
- y is the intended output of x_N

- Nothing is known about intention for middle layers (a.k.a. hidden units)
- Apply same error function
- And gradient descent

Multi Layer Networks

- The error function

$$\text{Err} = \frac{1}{2}[y - x_N]^2$$

- Gradient update:

$$w_{j,i} = w_{j,i} - \eta \frac{\partial \text{Err}}{\partial w_{j,i}}$$

- How can we calculate the gradient for an arbitrary $w_{j,i}$ (at middle or top layer)?

Multi Layer Networks

- The error function

$$\text{Err} = \frac{1}{2}[y - x_N]^2$$

- Gradient update:

$$w_{j,i} = w_{j,i} - \eta \frac{\partial \text{Err}}{\partial w_{j,i}}$$

- Two basic observations:

$$\frac{\partial \text{Err}}{\partial w_{j,i}} = \frac{\partial \text{Err}}{\partial s_i} \frac{\partial s_i}{\partial w_{j,i}}$$

$w_{j,i}$ affects output only through s_i

$$\frac{\partial s_i}{\partial w_{j,i}} = \frac{\partial \sum_j w_{j,i} x_j}{\partial w_{j,i}} = x_j$$

Just a derivative of linear function

Multi Layer Networks

- The error function

$$\text{Err} = \frac{1}{2}[y - x_N]^2$$

- Gradient update:

$$w_{j,i} = w_{j,i} - \eta \frac{\partial \text{Err}}{\partial w_{j,i}}$$

- Two basic observations:

$$\frac{\partial \text{Err}}{\partial w_{j,i}} = \frac{\partial \text{Err}}{\partial s_i} \frac{\partial s_i}{\partial w_{j,i}}$$

$$\frac{\partial \text{Err}}{\partial w_{j,i}} = \frac{\partial \text{Err}}{\partial s_i} \frac{\partial s_i}{\partial w_{j,i}} = \Delta_i x_j$$

$$\frac{\partial s_i}{\partial w_{j,i}} = \frac{\partial \sum_j w_{j,i} x_j}{\partial w_{j,i}} = x_j$$

Backpropagation Algorithm

- A few more steps (on the board) yield the Backpropagation algorithm
- Start by initializing all $w_{j,i}$ to small random values

Backpropagation Algorithm

- Repeat for each example (x,y):
  - For all i, compute values s_i and x_i by going forward in network

$$s_i = \sum_j w_{j,i} x_j$$
$$x_i = \sigma(s_i) = \sigma(\sum w_{j,i} x_j)$$

  - For all i, compute values Δ_i by going backward in network

$$\Delta_N = -(y - x_N) x_N (1 - x_N)$$
$$\Delta_i = x_i (1 - x_i) \sum_k \Delta_k w_{i,k}$$

  - Update all weights $w_{j,i} = w_{j,i} - \eta \Delta_i x_j$

Backpropagation Algorithm

- Algorithm on previous slide updates after each example
- This is known as "stochastic gradient descent" (similar to perceptron)
- The standard Backpropagation algorithm makes multiple iterations over training set: in each iteration it collects the gradients from all examples in the training set and only then makes an update.

Multi Layer Networks

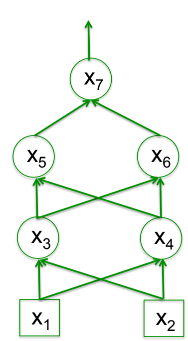


Illustration of Backpropagation

$$\eta = 0.1$$
$$w_{56} = w_{67} = 1$$
$$w_{35} = w_{36} = w_{45} = w_{46} = 0.6$$
$$w_{13} = w_{14} = w_{23} = w_{24} = 1$$

Input example : $(x_1, x_2) = (2, 3)$

Desired output : $L = 0$

Backpropagation Example

First Step: compute $s_i, x_i,$ and $\sigma'_i = x_i(1 - x_i)$

$s_3 = 1 * 2 + 1 * 3 = 5$ $\quad x_3 = \frac{1}{1+e^{-5}} = 0.993$ $\quad \sigma'_3 = 0.007$

$s_4 = 5$ $\quad x_4 = 0.993$ $\quad \sigma'_4 = 0.007$

$s_5 = 0.6 * 0.993 + 0.6 * 0.993 = 1.192$ $\quad x_5 = 0.767$ $\quad \sigma'_5 = 0.179$

$s_6 = 1.192$ $\quad x_6 = 0.767$ $\quad \sigma'_6 = 0.179$

$s_7 = 1 * 0.767 + 1 * 0.767 = 1.534$ $\quad x_7 = \frac{1}{1+e^{-1.534}} = 0.823$ $\quad \sigma'_6 = 0.146$

Backpropagation Example

Second Step: compute Δ_i

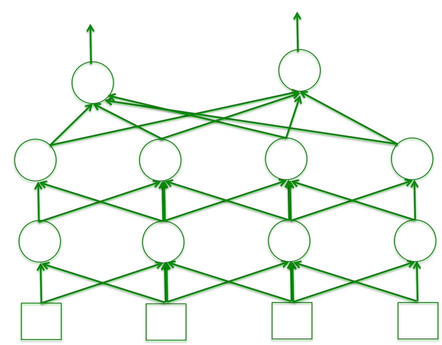
$$\Delta_7 = -\sigma'_7 * (L - x_7) = -0.146 * (0 - 0.823) = 0.120$$
$$\Delta_5 = \sigma'_5 w_{57} \Delta_7 = 0.179 * 1 * 0.120 = 0.021$$
$$\Delta_6 = \Delta_5$$
$$\Delta_3 = \sigma'_3 [w_{35}\Delta_5 + w_{36}\Delta_6] = 0.000176$$
$$\Delta_4 = \Delta_3$$

Backpropagation Example

Third Step: update weights

$$w_{13} = w_{13} - \eta x_1 \Delta_3 = 1 - 0.1 * 2 * 0.000176 = 0.9999648$$
$$\ldots$$
$$w_{35} = w_{35} - \eta x_3 \Delta_5 = 0.6 - 0.1 * 0.993 * 0.021 = 0.5579$$
$$\ldots$$
$$w_{57} = w_{57} - \eta x_5 \Delta_7 = 1 - 0.1 * 0.767 * 0.120 = 0.9908$$
$$\ldots$$

Multiple Output Nodes



Multiple Output Nodes

- All outputs share the same hidden layer
- Network identifies useful representations that are useful for all outputs
- Exactly same algorithm applies
- Forward pass identical
- Backward pass: each output unit calculates Delta using Δ_N formula

Multi Layer Networks

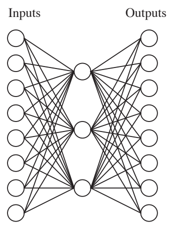
- Not easy to optimize; the error surface has a lot of local minima
- Solutions:
- Momentum:

$$w_{j,i} = w_{j,i} - \eta[\frac{\partial \text{Err}}{\partial w_{j,i}} + \alpha \text{ previous update}]$$

- Use multiple restarts and pick one with lowest training set error
- … many more recent techniques

What does the hidden layer do?

- Example: self-encoders

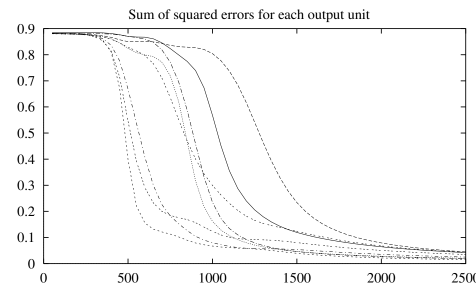


Learned hidden layer representation:

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

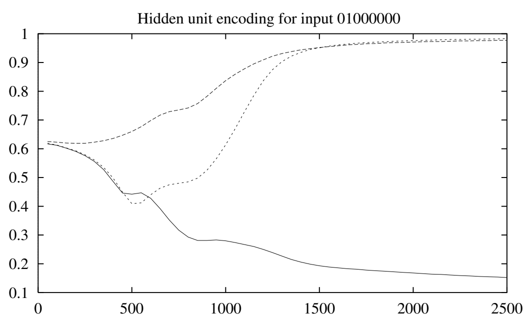
[Images from Mitchell's textbook]

What does the hidden layer do?



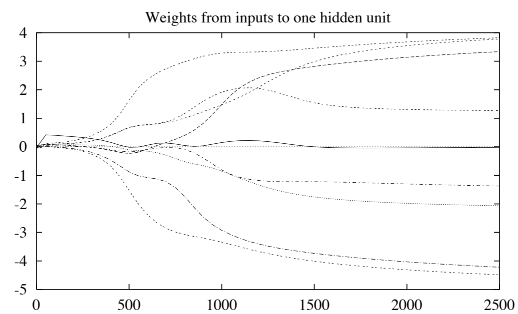
[Images from Mitchell's textbook]

What does the hidden layer do?



[Images from Mitchell's textbook]

What does the hidden layer do?

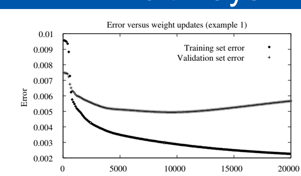


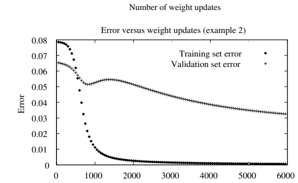
[Images from Mitchell's textbook]

Multi Layer Networks

- How to pick network size (and shape)?
- Similar to model selection in other models
  - cross validation
  - Combine fit + penalty
- How many updates?
  - Overfitting with large number of updates
  - Can do with with large network and moderate number of updates

Multi Layer Networks





Using validation set for stopping criterion per number of updates

[Images from Mitchell's textbook]

Multi Layer Networks

- Renewed interest in Deep Networks in last decade
- Several schemes for special network structure and special node functions
- Several schemes for training
- Combination of these ideas with BigData
- Yields
- Impressive improvements in performance in vision and other applications

Convolutional Networks

- Architecture inspired by vision system
- Alternating layers of grid based structures
- Each node calculates local function on patch from previous layer

Convolutional Networks

- Alternate layers of:
- "convolution layer" applies filter to patch from previous layer; weights repeat in all nodes (i.e. same filter)
- "Pooling layer" combines multiple filters of same block
- Followed by fully connected layers

Deep Networks

- Autoencoders: similar to 8-3-8 idea.
- Network fragments can be used to learn one level of internal representations in an unsupervised manner
- Restricted Boltzmann Machines (RBM): a probabilistic model with similar intuitive role
- Stacking these gives a deep network
- Further supervised training of entire model after this step

Deep Networks

- Active area of research
- Still not well understood
- Public interest due to empirical success
- Source of success: Huge data? Network architecture? Training algorithms? Domain specific engineering?

Multi Layer Neural Networks

- Complex representation of functions
- Can be trained with gradient based methods
- But training can be tricky
- Hidden layer "learning representation"
- Recent work on deep networks adds special architecture and/or training procedures