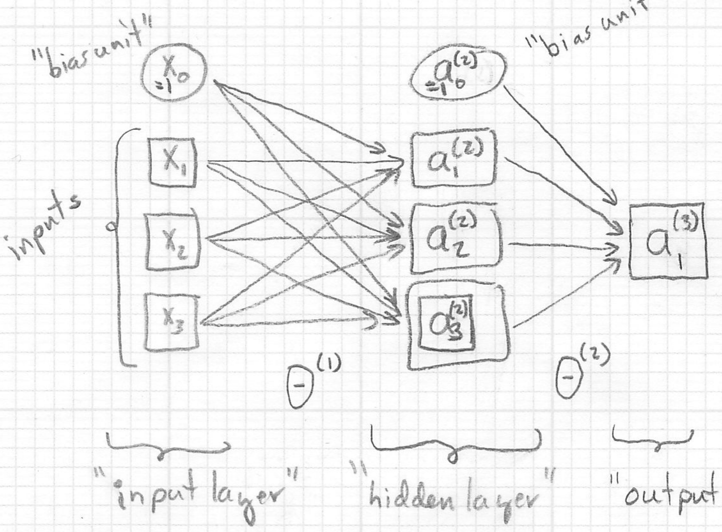
Neural Networks: Representation



$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3)$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3)$$
$$a_1^{(3)} = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11} a_1^{(2)} + \dots)$$

$$a^{(j+1)} = g(\Theta^T a^{(j)}) = g(z^{(j+1)}) \qquad a^{(1)} = x, \quad z^{(j+1)} = \Theta^T a^{(j)}$$

Note that input data usually resides in rows of an input matrix:

$$X = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} \dots & x_n^{(1)} \\ & & & \\ 1 & x_1^{(m)} & x_2^{(m)} \dots & x_n^{(m)} \end{bmatrix}$$

If $\Theta^{(j)}$ is stored in-order, then it must be transposed to conform with X, and the resulting output vector $a^{(j+1)}$ is also a row vector:

$$X \cdot \Theta^{(1)T} = [x_0 \; x_1 \; x_2 \dots] \begin{bmatrix} \Theta_{10}^{(1)} & \Theta_{20}^{(1)} & \dots \\ \Theta_{11}^{(1)} & \Theta_{21}^{(1)} & \dots \\ \Theta_{12}^{(1)} & & \ddots \\ \vdots & & \end{bmatrix}$$
$$= [z_1^{(2)} \; z_2^{(2)} \; z_3^{(2)} \dots] = z^{(2)}$$

Then $a^{(2)} = g(z^{(2)})$ is in order for multiplication with $\Theta^{(2)T}$ and so on down the chain.

For classification purposes, we select the output layer element with the highest activation level. Octave provides built-in functions for this step.

Neural Networks: Cost Function (lecture 9)

20120215 ABR ②

Sum over output units

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1-y_k^{(i)})\log(1-(h_\Theta(x^{(i)}))_k)\right]$$

$$+ \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}} \left(\Theta_{ji}^{(l)}\right)^2$$

Where: $h_\Theta(x) \in \mathbb{R}^K$, $(h_\Theta(x))_i = i^{th}$ output

K = number of classes, number of output units ($K = s_L$)

s_l = number of units in layer l (not counting bias unit)

L = total number of layers in network

$y \in \{0,1\}^K$

Back-Propagation: Gradient and Algorithm

need: $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)}$ (ignoring λ for now)

$\delta_j^{(l)}$ = "error" of node j in layer l

$\delta_j^{(L)} = a_j^{(L)} - y_j \Rightarrow \delta^{(L)} = a^{(L)} - y$

$\delta^{(L-1)} = \Theta^{(L-1)T}\delta^{(L)} .* g'(z^{(L-1)}) \overset{?}{\Rightarrow} \boxed{g'(z^{(L-1)}) = a^{(L-1)} .* (1-a^{(L-1)})}$

$\delta^{(L-2)} = \Theta^{(L-2)T}\delta^{(L-1)} .* g'(z^{(L-2)})$

$\vdots$

No $\delta^{(1)}$

Algorithm:

- set $\Delta_{ij}^{(l)} = 0$
- for $i = 1$ to m:
  - $a^{(1)} = x^{(i)}$
  - perform forward propagation to compute $a^{(l)}$ for $l = 2, \ldots, L$
  - using $y^{(i)}$, compute $\delta^{(L)} = a^{(L)} - y^{(i)}$
  - for $l = L-1$ to 2:
    $\delta^{(l)} = \Theta^{(l)T}\delta^{(l+1)} .* g'(z^{(l)})$
  - $\Delta_{ij}^{(l)} \mathrel{+}= a_j^{(l)}\delta_i^{(l+1)} \rightarrow \boxed{\Delta^{(l)} = \Delta^{(l)} + \delta^{(l+1)} \cdot a^{(l)T}}$

Neural Network: Back-propagation (continued)

20120215 ABR

regularization terms are added as follows:

$$D_{ij}^{(l)} = \frac{1}{m}\Delta_{ij}^{(l)} + \lambda\Theta_{ij}^{(l)} \qquad j \neq 0$$

$$D_{ij}^{(l)} = \frac{1}{m}\Delta_{ij}^{(l)} \qquad j = 0$$

Intuition:

consider the case of one training example (x, y)

$a^{(1)} = x$

$z^{(2)} = \Theta^{(1)} a^{(1)}$

$a^{(2)} = g(z^{(2)})$ (don't forget to add $a_0^{(2)}$)

$z^{(3)} = \Theta^{(2)} a^{(2)}$

$a^{(3)} = g(z^{(3)})$ $\qquad a_0^{(3)}$

$z^{(4)} = \Theta^{(3)} a^{(3)}$

$a^{(4)} = h_\Theta(x) = g(z^{(4)})$

or,

$$h_\Theta(x) = g(\Theta^{(3)} g(\Theta^{(2)} g(\Theta^{(1)} \cdot x)))$$

Back-Propagation: Concise Summary

- Set $\Delta_{ij}^{(l)} = 0$
- For $i = 1$ to m:
  - $a^{(1)} = x^{(i)}$
  - compute all layer activations $a^{(l)}$
  - $\delta^{(L)} = a^{(L)} - y^{(i)}$
  - for $l = L-1$ to 2:
    - $\delta^{(l)} = \Theta^{(l)T} \delta^{(l+1)} .* (a^{(l)} .* (1 - a^{(l)}))$
  - $\Delta_{ij}^{(l)} \mathrel{+}= a_j^{(l)} \delta_i^{(l+1)}$ or $\Delta^{(l)} = \Delta^{(l)} + \delta^{(l+1)} \cdot a^{(l)T}$
- $D_{ij}^{(l)} = \frac{1}{m}\Delta_{ij}^{(l)} + \lambda\Theta_{ij}^{(l)}$ $\qquad j \neq 0$
- $D_{ij}^{(l)} = \frac{1}{m}\Delta_{ij}^{(l)}$ $\qquad j = 0$

← These are the components of the gradient

$$\boxed{\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}}\left[y^{(i)}\log(h_\Theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\Theta(x^{(i)}))\right]}$$

* Use $\left[\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) \approx \frac{J(\Theta+\epsilon) - J(\Theta-\epsilon)}{2\epsilon}\right]$ to test gradient function from back propagation.

** constant-initialization ("problem of symmetric weights") does not work ($\frac{\partial}{\partial\Theta} J = 0$)

"Problem of Symmetric Weights"

Constant initialization is a "flat" spot on the cost function. Initialize with random deviates to break symmetry.

Network Architecture Choices

$\left.\begin{array}{l}\text{\#Input units}\\ \text{\#Output units}\end{array}\right\}$ defined by problem

hidden layers → 1 is a reasonable default

units/hidden layer → generally more is better

Training

1. Randomly initialize weights
2. Implement forward propagation to get $h_\Theta(x^{(i)})$ for any $x^{(i)}$
3. Implement code to compute cost function $J(\Theta)$
4. Implement back-propagation to compute partial derivatives $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$

for **i = 1 : m** (this can be vectorized)
  Perform forward propagation & backward propagation using example $(x^{(i)}, y^{(i)})$, saving activations and deltas $(a^{(l)}, \delta^{(l)})$ for $l = 2, \dots, L$
  Compute Δ
Compute $D(\Theta, \lambda)$

5. Use gradient checking to test backpropagation code
6. Use gradient descent or optimized solvers to minimize $J(\Theta)$