Informal Ng machine learning class notes PhL last update: 5.20.12a

This is a "live" table of contents:

This file contains three sets of *very crude* notes that were thrown together (probably riddled with errors) with regard to Ng's Stanford machine learning course. I think these notes derive most everything that Andrew Ng omitted from his lectures due to the mathematics involved (but only through video 9-8). These notes do assume a knowledge of elementary calculus. I know nothing about machine learning except what was presented in the Ng videos through video 9-8.

Notes #1: Computes the exact solution of two fitting problems which are pretty much identical, and derives the "normal equation" solution $\boldsymbol{\theta} = (X^T X)^{-1} X^T \mathbf{y}$ stated by Mr. Ng.

Notes #2: Provides some elementary support for the cost function in terms of norms and metrics, then discusses both the analog and digital fits with vector outputs instead of scalar outputs. Ng's digital case cost function is derived for a vector output. This vector digital model (logistical regression) is later used in the backprop notes #3. Along the way, the "normal equation" is found for the analog case with a vector output and seems to be $\theta^T = (X^T X)^{-1} X^T Y$.

Notes #3: This is a derivation of the backprop algorithm, soup to nuts.

If you find errors in these notes and report them to me, I will update the notes with your corrections.
-- Phil Lucht

rimrock@xmission.com

Notes #1 on Ng machine learning PhL 5.9.12

There are two different data fitting problems that have basically the same solution. Here I use a_i instead of θ_i for parameters and I use "trial set" for "training set".

Problem 1: Take an m-element data set (trial set 1,2..m) with *one feature* $x^{(i)}$ (where i = 1..m labels elements of the trial set) and find the best *polynomial fit* of degree $n \le m$. Write this polynomial (generic argument x) as

$$p_n(x) = \Sigma_{j=0}^{n} a_j x^j = a_0 + a_1 x + a_2 x^2 + ... + a_n x^n$$

where there are n +1 constants a_j we seek to determine. For example, at trial point $x^{(i)}$ the above says

$$p_n(x^{(i)}) = \Sigma_{j=0}^{n} a_j [x^{(i)}]^j = a_0 + a_1 x^{(i)} + a_2 [x^{(i)}]^2 + ... + a_n [x^{(i)}]^n \quad .$$

For n = 1 this would be a linear fit, for n = 2 quadratic, for n = 3 cubic and so on.

The problem is to find the constants a_j that minimize this least squares fit cost function,

$$J(\mathbf{a}) = \Sigma_{i=1}^{m} (y^{(i)} - p_n(x^{(i)}))^2 \qquad \mathbf{a} = (a_0, a_1....a_n)$$

$$= \Sigma_{i=1}^{m} (y^{(i)} - \Sigma_{j=0}^{n} a_j [x^{(i)}]^j)^2$$

I will solve this problem below. Note that $[x^{(i)}]^j$ is the number $x^{(i)}$ raised to the j^th power.

Problem 2: Take an m-element data set (trial set 1,2..m) with *n features* $x^{(i)}_j$ (where i = 1..m labels elements of the trial set and j labels the features 1 to n) and find the best *linear fit*. Write this linear fit (generic argument x) as [now $p_n(\mathbf{x})$ is a function of $\mathbf{x}$, not a polynomial]

$$p_n(\mathbf{x}) = a_0 + a_1 x_1 + a_1 x_2 + ... + a_n x_n \qquad // \text{ quadratic terms would be like } b_{ij} x_i x_j ...$$

For example, at trial point $\mathbf{x}^{(i)}$ the above says

$$p_n(\mathbf{x}^{(i)}) = a_0 + a_1 x^{(i)}_1 + a_1 x^{(i)}_2 + ... + a_n x^{(i)}_n \quad .$$

Define $x^{(i)}_0 \equiv 1$ for all trial set members i (this is a "dummy feature") so then

$$p_n(\mathbf{x}^{(i)}) = a_0 x^{(i)}_0 + a_1 x^{(i)}_1 + a_1 x^{(i)}_2 + ... + a_n x^{(i)}_n$$

$= \Sigma_{j=0}^{n} a_j x^{(i)}_j = \mathbf{a} \bullet \mathbf{x}^{(i)}$ $\mathbf{a} = (a_0, a_1....a_n)$ $\mathbf{x}^{(i)} = (x^{(i)}_0, x^{(i)}_1, ...)$

which can also be written as $\mathbf{a} \bullet \mathbf{x}^{(i)} = \mathbf{a}^T \mathbf{x}^{(i)}$. The problem is to find the a_j that minimize this cost function,

$$J(\mathbf{a}) = \Sigma_{i=1}^{m} (y^{(i)} - p_n(\mathbf{x}^{(i)}))^2$$

$$= \Sigma_{i=1}^{m} (y^{(i)} - \Sigma_{j=0}^{n} a_j x^{(i)}_j)^2$$

Note the similarity of this form to that of Problem 1. Note that $x^{(i)}_j$ is the value of the j^{th} feature for trial point i.

Relation between these two problems. Suppose you solve Problem 1 for the polynomial coefficients a_i which minimize the cost function. If in this solution you replace $[x^{(i)}]^j \rightarrow x^{(i)}_j$, then you will arrive at the solution of Problem 2!

Solution of Problem 1

To find a minimum of J, we set all the partial derivatives with respect to the a_i to zero. I use the notation that $\partial/\partial a_k$ (partial derivative with respect to a_k) and then for short, $\partial_{ak} \equiv \partial/\partial a_k$. Then

$$J(\mathbf{a}) = \Sigma_{i=1}^{m} (y^{(i)} - \Sigma_{j=0}^{n} a_j [x^{(i)}]^j)^2 \qquad \text{// least squares fit cost function}$$

$$\partial_{ak} J(\mathbf{a}) = \Sigma_{i=1}^{m} 2 (y^{(i)} - \Sigma_{j=0}^{n} a_j [x^{(i)}]^j)\ \partial_{ak} (y^{(i)} - \Sigma_{j=0}^{n} a_j [x^{(i)}]^j) \qquad \text{// chain rule}$$

$$= \Sigma_{i=1}^{m} 2 (y^{(i)} - \Sigma_{j=0}^{n} a_j [x^{(i)}]^j)\ (- \Sigma_{j=0}^{n} \{\partial_{ak} a_j\} [x^{(i)}]^j)$$

$$= \Sigma_{i=1}^{m} 2 (y^{(i)} - \Sigma_{j=0}^{n} a_j [x^{(i)}]^j)\ (- \Sigma_{j=0}^{n} \{\delta_{k,j}\} [x^{(i)}]^j) \qquad \text{// } \partial a_j/\partial a_k = \delta_{k,j}$$

$$= \Sigma_{i=1}^{m} 2 (y^{(i)} - \Sigma_{j=0}^{n} a_j [x^{(i)}]^j)\ (- [x^{(i)}]^k) .$$

Then set $\partial_{ak} J(\mathbf{a}) = 0$ for all k = 0,1...n. We can then forget the 2 and minus sign to get

$$\Sigma_{i=1}^{m} (y^{(i)} - \Sigma_{j=0}^{n} a_j [x^{(i)}]^j)\ [x^{(i)}]^k = 0$$

or

$$(\Sigma_{i=1}^{m} y^{(i)} [x^{(i)}]^k) - \Sigma_{j=0}^{n} a_j (\Sigma_{i=1}^{m} [x^{(i)}]^j [x^{(i)}]^k) = 0 .$$

These are called "**the normal equations**". Given the dataset $\{ x^{(i)}, y^{(i)}\}$, we can go compute these quantities,

$$t_k \equiv \Sigma_{i=1}^{m} y^{(i)} [x^{(i)}]^k$$

$$S_{kj} \equiv \Sigma_{i=1}^{m} [x^{(i)}]^j [x^{(i)}]^k . \qquad \text{// S is a symmetric matrix, } S_{kj} = S_{jk}$$

Then the normal equations can be written as

$t_k - \Sigma_{j=0}^n a_j S_{kj} = 0$ for k = 0,1,2...n,

or

$\Sigma_{j=0}^n S_{kj} a_j = t_k$. // there are n+1 normal equations in n+1 unknowns a_j

But this is a matrix equation of the form (S = matrix, **a** and **t** are column vectors)

$S\,\mathbf{a} = \mathbf{t}$ // S has n+1 rows and n+1 columns so is "square"

and the solution of this equation is seen to be (that is, apply S^{-1} from the left on both sides)

$\mathbf{a} = S^{-1}\mathbf{t}$

where S^{-1} is the inverse of matrix S given by $S^{-1} = \mathrm{cof}(S^T)/\det(S)$. Problem is solved (unless det(S)=0).

Solution of Problem 2

Go through all the same motions as for the solution of Problem 1 with the change $[x^{(i)}]^j \rightarrow [x^{(i)}]_j$. Thus, the solution to Problem 2 is given by

$\mathbf{a} = S^{-1}\mathbf{t}$ $S^{-1} = \mathrm{cof}(S^T)/\det(S)$

where

$t_k \equiv \Sigma_{i=1}^m y^{(i)} [x^{(i)}]_k$.

$S_{kj} \equiv \Sigma_{i=1}^m [x^{(i)}]_j [x^{(i)}]_k$ // S is a symmetric matrix, $S_{kj} = S_{jk}$

To make this look like what Ng states in his lecture, define the following design matrix X,

$X_{ik} \equiv [x^{(i)}]_k$ i = sample k = feature // X has m rows and n+1 columns, not square

Then one has

$S_{jk} = S_{kj} \equiv \Sigma_{i=1}^m [x^{(i)}]_j [x^{(i)}]_k = \Sigma_{i=1}^m X_{ij} X_{ik} = \Sigma_{i=1}^m X^T{}_{ji} X_{ik} = (X^TX)_{jk}$

$\Rightarrow S = X^TX$ // this combination of two non-square matrices makes a square one

$t_k = \Sigma_{i=1}^m y_i [x^{(i)}]_k = \Sigma_{i=1}^m y_i X_{ik} = \Sigma_{i=1}^m X^T{}_{ki} y_i = (X^T\mathbf{y})_k$

$\Rightarrow \mathbf{t} = X^T\mathbf{y}$

Then the solution is given by

$\mathbf{a} = S^{-1}\mathbf{t} = (X^TX)^{-1} X^T\mathbf{y}$

In Maple or Octave or MATLAB this is a single instruction basically and the result can be made as accurate as required by setting the representation decimal count to as large as you want.

Note: The determinant of a matrix vanishes if any row (column) is a multiple of another row (column). I think this is why there are problems if two sets of features are basically representations of the same feature set. DetS = 0 means S is not invertible since $S^{-1} = \mathrm{cof}(S^T)/\det(S)$.

Notes #2 on Ng machine learning PhL 5.17.12

1. Review of the Linear Regression (analog) case.

wiki: " The term "regression" was coined by Francis Galton in the nineteenth century to describe a biological phenomenon. The phenomenon was that the heights of descendants of tall ancestors tend to regress down towards a normal average (a phenomenon also known as regression toward the mean).[7][8] "

The analog least squares linear fit cost (error) function was taken to be (now replace $\mathbf{a} \rightarrow \boldsymbol{\theta}$ as in Ng, and add a factor 1/2m in front as Ng does)

$$J(\boldsymbol{\theta}) = (1/2m)\, \Sigma_{i=1}{}^{m}\, [\, h_\theta(\mathbf{x}^{(i)}) - y^{(i)}]^2$$

$$= (1/2m)\Sigma_{i=1}{}^{m}\, [\Sigma_{j=0}{}^{n}\, \theta_j\, x^{(i)}{}_j - y^{(i)}]^2 \qquad h_\theta(\mathbf{x}) = \boldsymbol{\theta} \bullet \mathbf{x} = \Sigma_j\theta_j x_j$$

$$= (1/2m)\Sigma_{i=1}{}^{m}\, [\boldsymbol{\theta}\bullet\, \mathbf{x}^{(i)} - y^{(i)}\,]^2$$

from which we (previous doc) got the normal equation finding the solution value for $\boldsymbol{\theta}$

$$\boldsymbol{\theta} = (X^TX)^{-1} X^T\mathbf{y} \qquad \text{where } X_{ik} \equiv [x^{(i)}]_k \qquad y_i = y^{(i)} \qquad \boldsymbol{\theta} = (\theta_0,\theta_1 \ldots \theta_n)$$

and the derivatives for gradient descent were (adding the 1/2m compared to earlier notes)

$$\partial J(\boldsymbol{\theta})/\partial\theta_j = -(1/m)\, \Sigma_{i=1}{}^{m}\, (y^{(i)} - \Sigma_{j=0}{}^{n}\, \theta_j\, [x^{(i)}]^j\,)\, [x^{(i)}]_j$$

$$= (1/m)\, \Sigma_{i=1}{}^{m} (\, \Sigma_{j=0}{}^{n}\, \theta_j\, [x^{(i)}]^j - y^{(i)})\ [x^{(i)}]_j$$

$$= (1/m)\, \Sigma_{i=1}{}^{m} (\, \boldsymbol{\theta}\bullet\, \mathbf{x}^{(i)} - y^{(i)})\ [x^{(i)}]_j$$

$$= (1/m)\, \Sigma_{i=1}{}^{m} (h_\theta(\mathbf{x}^{(i)}) - y^{(i)})\ [x^{(i)}]_j$$

or in vector notation ($\nabla^{(\theta)}$ is the "gradient operator" with respect to variables θ_i)

$$\nabla^{(\theta)} J(\boldsymbol{\theta}) = (1/m)\, \Sigma_{i=1}^{m} (h_\theta(\mathbf{x}^{(i)}) - y^{(i)})\, \mathbf{x}^{(i)} \ .$$

In regular x,y space, the gradient vector $\nabla f(x,y)$ always points "up hill" on the 3D surface $z = f(x,y)$, so that is why there is a minus sign in the iteration step of gradient descent -- you want to go "down hill".

In the above, the "best linear fit" to the data, based on minimizing the sum of the squares of the differences between the training points and the linear fit, is given by this linear modeling function

$$h_\theta(\mathbf{x}) = \boldsymbol{\theta} \bullet \mathbf{x} \quad = \boldsymbol{\theta}^T\mathbf{x} \ = \mathbf{x}^T\boldsymbol{\theta} \quad = \theta_0 x_0 + \theta_1 x_1 + ...$$

which Ng likes to call a "**hypothesis**", but we would just call it a "fit to the data" or "model for the data".
This function h_θ is "linear" in **x**, which means exactly these two facts:

$h_\theta(\mathbf{x} + \mathbf{z}) = h_\theta(\mathbf{x}) + h_\theta(\mathbf{z})$ // since $\boldsymbol{\theta}\bullet (\mathbf{x} + \mathbf{z}) = \boldsymbol{\theta}\bullet\mathbf{x} + \boldsymbol{\theta}\bullet\mathbf{z}$

$h_\theta(\alpha\mathbf{x}) = \alpha\, h_\theta(\mathbf{x})$ // since $\boldsymbol{\theta}\bullet(\alpha\mathbf{x}) = \alpha\, \boldsymbol{\theta}\bullet\mathbf{x}$

2. Norms, Metrics and Dot Products

The vector space R^n has elements like **x** and **y** which can be added and subtracted and stretched

$\mathbf{x} + \mathbf{y} \qquad \mathbf{x} - \mathbf{y} \qquad \alpha\mathbf{x} \qquad (\alpha = \text{real}) \qquad \mathbf{x} = (x_1, x_2, ...x_n)$

One often defines a set of unit basis vectors in a vector space, call them $\hat{\mathbf{u}}_i$, which point along the axes of the space. In a Cartesian coordinate system, these axes are orthogonal to each other. One notation commonly used is $\hat{\mathbf{u}}_1 = \hat{\mathbf{x}}$, $\hat{\mathbf{u}}_2 = \hat{\mathbf{y}}$, $\hat{\mathbf{u}}_3 = \hat{\mathbf{z}}$, but the general notation $\hat{\mathbf{u}}_i$ works just fine. In terms of such unit vectors one can write a vector as

$$\mathbf{x} = \Sigma_i\, x_i\, \hat{\mathbf{u}}_i$$

where the components x_i are then the projections on to each axis of the vector **x**.
The particular vector space R^n also has the notion of the "length of a vector" $||\ \mathbf{x}\ ||$ which is this:

$||\ \mathbf{x}\ ||^2 = \ \Sigma_i x_i^2 \ = \Sigma_i\, |x_i|^2$ $\qquad$ $|\ a\ |$ means absolute value of a real number

where $||\ \mathbf{x}\ ||$ is called "the norm of vector **x**". A norm is always positive and is zero when the vector is set to 0. This particular norm is called the **L_2 norm** since power 2 appears in $\Sigma_i\, |x_i|^2$. Thus, R^n is not just a vector space, it is also a normed space (sometimes called a Banach space). Using this norm, one can then define a notion of "the distance $d(\mathbf{x},\mathbf{y})$ between two vectors" as follows

$$d^2(\mathbf{x},\mathbf{y}) = ||\ \mathbf{x} - \mathbf{y}\ ||^2 \ = \Sigma_i(x_i\text{-}y_i)^2$$
$$d(\mathbf{x},\mathbf{y}) \ = ||\ \mathbf{x} - \mathbf{y}\ || \ = \sqrt{\Sigma_i(x_i\text{-}y_i)^2}$$

and any such distance function d($\mathbf{x}$,$\mathbf{y}$) is called a **metric**. With this metric, our space is a vector space, a normed space, and a metric space! A metric should always be positive and should be zero when the two vectors are equal. Defining the metric in terms of the norm as above guarantees these properties, though other metric definitions are possible.

The vector space R^n finally has another property called the **inner product** or **scalar product** or **dot product**. It is defined as

$$\mathbf{x} \bullet \mathbf{y} = \Sigma_i x_i y_i$$

Thus we see that

$$\| \mathbf{x} \|^2 = \Sigma_i x_i^2 = \mathbf{x} \bullet \mathbf{x}$$

$$\| \mathbf{x\text{-}y} \|^2 = \Sigma_i (x_i - y_i)^2 = (\mathbf{x\text{-}y}) \bullet (\mathbf{x\text{-}y}) = d^2(\mathbf{x},\mathbf{y})$$

A space with a norm, a metric, and a dot product is called a **Hilbert Space** and R^n is a Hilbert Space.

The dot product can be written in **matrix notation** as follows, where $\mathbf{y}$ is a column vector and $\mathbf{x}$ is a column vector but $\mathbf{x}^T$ (transpose) is a row vector:

$$\mathbf{x} \bullet \mathbf{y} = \Sigma_i x_i y_i = (x_1, x_2 \ldots. x_n) \begin{pmatrix} y_1 \\ y_2 \\ \ldots \\ y_n \end{pmatrix} = \mathbf{x}^T\mathbf{y} = x_1y_1 + x_2y_2 + \ldots x_ny_n$$

It should be pretty obvious that

$$\mathbf{x} \bullet \mathbf{y} = \mathbf{y} \bullet \mathbf{x} = \mathbf{x}^T\mathbf{y} = \mathbf{y}^T\mathbf{x} \quad .$$

In R^3 it is not hard to show that

$$\mathbf{x} \bullet \mathbf{y} = \| \mathbf{x} \| \, \| \mathbf{y} \| \cos\theta$$

where θ is the angle between the two vectors $\mathbf{x}$ and $\mathbf{y}$. Thus, vectors which are perpendicular have a zero dot product since $\cos(\pi/2)$ = cos(90 degrees) = 0. Using the unit basis vectors as examples, we can find the following

$\hat{\mathbf{u}}_i \bullet \hat{\mathbf{u}}_i = \| \hat{\mathbf{u}}_i \|^2 = 1$ // since a "unit vector" has $\| \hat{\mathbf{u}}_i \| = 1$

$\hat{\mathbf{u}}_i \bullet \hat{\mathbf{u}}_j = 0$ if $i \neq j$ // since the Cartesian axes are perpendicular

so

$\hat{\mathbf{u}}_i \bullet \hat{\mathbf{u}}_j = \delta_{i,j}$ // **Kronecker delta** = 1 if i=j, = 0 if $i \neq j$.

The norm and metric and inner (scalar, dot) product stated above are the official ones which define the Hilbert Space known as R^n. Other norms and other metrics are also possible. The most famous alternative is the L_1 norm defined as

$$|| \mathbf{x} || = \Sigma_i |x_i| .$$

If you think of each $|x_i|$ as the length of a block in NYC (R^2) then this norm is the distance a taxicab drives to get from location 0 to location $\mathbf{x}$. (Manhattan norm, taxi-cab norm). More generally, the L^p norm is defined by $|| \mathbf{x} || = \Sigma_i |x_i|^p$. Of course for each of these alternative norms, you can define an alternative distance metric according to $d(\mathbf{x},\mathbf{y}) = || \mathbf{x} - \mathbf{y} ||$, hence the Manhattan distance, for example.

If n = 1, we can talk about the space R^1 as a special case of R^n . Then one has $x_1 = x$ and

$$|| x ||^2 = x^2 = |x|^2$$

$$d^2(x,y) = || x - y ||^2 = (x-y)^2$$
$$d(x,y) = || x - y || = \sqrt{(x-y)^2}$$

3. Another review of the Ng analog linear case

$$J(\boldsymbol{\theta}) = \Sigma_{i=1}^m [h_\theta(\mathbf{x}^{(i)}) - y^{(i)}]^2 = \Sigma_i d^2(h_\theta(\mathbf{x}^{(i)}, y^{(i)})$$

Here we see that this cost function is based on the L^2 metric in R^1 just noted above,

$$d^2(h_\theta(\mathbf{x}^{(i)}, y^{(i)}) = [h_\theta(\mathbf{x}^{(i)}) - y^{(i)}]^2 ,$$

where i denotes the i^{th} training element. A fit using this L^2 metric is called a "least squares" fit. Here, $d(h_\theta(\mathbf{x}), y)$ is the "distance" between a value y and the model function (or hypothesis) $h_\theta(\mathbf{x})$ used as a fitting function to estimate y. Ng could have used the L^1 norm and written a different cost function

$$J(\boldsymbol{\theta}) = \Sigma_{i=1}^m | h_\theta(\mathbf{x}^{(i)}) - y^{(i)} | \qquad \text{// using the } L^1 \text{ norm}$$

but this causes the math to be uglier (y = |x| has a kink at x=0, eg, so not differentiable there) and that is one reason it is not typically used. Also, the L^2 norm has certain statistic properties which make it favored (I think).

Notice that the analog cost function $J(\boldsymbol{\theta})$ used in the least squares fit is the sum of the *squares* of the metric distances for each training set pair. One could have used some other power but using the square is what allows the simple normal function solution to be found. If one instead summed the metrics to the first power, one would have

$$J(\boldsymbol{\theta}) = \Sigma_{i=1}^m \sqrt{[h_\theta(\mathbf{x}^{(i)}) - y^{(i)}]^2} = \Sigma_i d(h_\theta(\mathbf{x}^{(i)}, y^{(i)})$$

and then the square root makes the math messier. This still would be a viable cost function for a gradient descent solution. In the digital case below, we will be summing the first power of the metrics since in that case it gives a simpler result.

4. The Ng digital fit (binary classification = logical = logistical regression) .

Apologies: In this and later sections I use the word "bit" sometimes to mean a binary number which is 0 or 1, and sometimes to mean a "soft bit" which is real number in the range (0,1) which you could round up or down to get the best estimate of it as a hard bit. In a neural network, one would presumably round the final outputs to actual bits (as in a digital demodulator, say), but the internal "bit lines" of the net carry soft bits which are later called a_k, the node activations.

When the $y^{(i)}$ outcomes can only be 0 or 1, Ng uses the following metric to represent the distance between a value and the model prediction (log here means natural log ln, but it could be any base log)

$$d(s,y) = -\log(s) \qquad \text{if } y = 1$$
$$d(s,y) = -\log(1-s) \qquad \text{if } y = 0 \ .$$

In the first line, one has $d(1,1) = -\log(1) = 0$, and $d(1,0) = -\log(0) = +\infty$.
In the second line, one has $d(0,0) = -\log(1) = 0$, and $d(0,1) = -\log(0) = +\infty$.

Notice that this choice of a metric satisfies the requirements of any metric that the distance be 0 when the two vectors are the same (again, these are scalar vectors = bits), and the distance is always positive. Also the distance is very large when the binary scalar numbers are exactly unequal. One could imagine other possible metrics, but this one is going to be good for calculation purposes.

This metric d(s,y) is specialized to the case that y = 0 or 1, while s can be any real number in the range $-1 \le s \le 1$ and in practice $-1 < s < 1$. For example

$$d(.99,1) = -\log(.99) = .004 \qquad \text{a very small cost or distance}$$
$$d(.01,1) = -\log(.01) = +2 \qquad \text{a large cost or distance}$$

$$d(.01,0) = -\log(.99) = .004 \qquad \text{a very small cost or distance}$$
$$d(.99,0) = -\log(.01) = +2 \qquad \text{a large cost or distance}$$

For $-1 < s < 1$ and y = 0 or 1, the metric is always positive, as required of any metric.

The two lines above can be trivially combined onto a single line

$$d(s,y) = -y \log(s) - (1-y) \log(1-s) \qquad \text{// valid for } y = 1 \text{ and } y = 0$$

Note that here s is just a dummy name for a variable which is an estimate of y. If we regard $s = h_\theta(\mathbf{x})$ as a model function for s, this metric is written

$$d(h_\theta(\mathbf{x}),y) = -y \log(h_\theta(\mathbf{x})) - (1-y) \log(1- h_\theta(\mathbf{x})) \ .$$

Now **x** is the vector of features in the model and $h_\theta(\mathbf{x})$ is some "model function" which one hopes is good to predict the training values in some average sense, and to predict new values as well. Ng uses the following model function

$h_\theta(\mathbf{x}) = g(\boldsymbol{\theta}\bullet\mathbf{x}) = \dfrac{1}{1+e^{-\boldsymbol{\theta}\bullet\mathbf{x}}}$ $\qquad$ $g(z) = \dfrac{1}{1+e^{-z}}$ = sigmoid function

If there are n+1 features $\mathbf{x} = (x_0=1, x_1, x_2....x_n)$ where x_0 is the "dummy feature" $x_0=1$, then there are n+1 components of vector $\boldsymbol{\theta} = (\theta_0, \theta_1...\theta_n)$.

The function g(z) maps the entire real axis into the range (-1,1) and thus meets the requirements noted above for the specialized metric d(g,y) where g must lie in (-1,1).

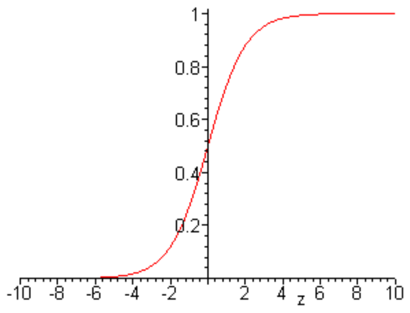
So in the analog regression model the linear model function was taken as $h_\theta(\mathbf{x}) = \boldsymbol{\theta} \bullet \mathbf{x}$, whereas in the digital regression model we shall use $h_\theta(\mathbf{x}) = g(\boldsymbol{\theta} \bullet \mathbf{x})$, where the extra function g(z) is used to get $h_\theta(\mathbf{x})$ into the required metric range (-1,1).

One could imagine other functions g(z), such as $(1+e^{-5z})^{-1}$ or $(2/\pi) \tan^{-1}(z)$, that could be used in place of the g(z) stated above. But the simple $g(z) = (1+e^{-z})^{-1}$ has a simple derivative, and is very smooth, and is anti-symmetric about z = 0, and is in general just a "nice choice".

```
g := 1/(1+exp(-z));
```

$$g := \frac{1}{1+e^{(-z)}}$$

```
plot(g, z= -10..10);
```



A Greek sigma Σ is the letter S and that is what this "sigmoid" function looks like.

There might be some theoretical arguments about why this g(z) might be close to optimal for the purpose at hand. Below we shall see some "properties of g(z)" which in fact give pretty amazing results for the functional form of the digital fit gradient.

Based on the above metric, the digital cost function is this (sum of trial metric error distances)

$$J(\boldsymbol{\theta}) = (1/m)\, \Sigma_{i=1}^{m}\; d(h_\theta(\mathbf{x}^{(i)}), y^{(i)})$$

$$= (1/m)\, \Sigma_{i=1}^{m}\; [-y^{(i)} \log(h_\theta(\mathbf{x}^{(i)})) - (1- y^{(i)}) \log(1- h_\theta(\mathbf{x}^{(i)}))\,]$$

$$= -\, (1/m)\, \Sigma_{i=1}^{m}\; [y^{(i)} \log(h_\theta(\mathbf{x}^{(i)})) + (1- y^{(i)}) \log(1- h_\theta(\mathbf{x}^{(i)}))\,]$$

$$= -\, (1/m)\, \Sigma_{i=1}^{m}\; [y^{(i)} \log(g(\boldsymbol{\theta} \bullet \mathbf{x}^{(i)})) + (1- y^{(i)}) \log(1- g(\boldsymbol{\theta} \bullet \mathbf{x}^{(i)}))\,]\; .$$

Ng has added the positive constant (1/m) as shown (m = number of training set elements) which of course makes no difference in a cost function but tends to normalize the value of J as m gets large.

The plan is to compute $\partial J/\partial\theta_j$ and set this to 0 to determine the θ_j from something like the normal equation, or if there is no simple such equation, then the $\partial J/\partial\theta_j$ are used in the gradient descent method.

5. Computation of derivatives for logistical regression

In the previous section we showed that,

$J(\boldsymbol{\theta}) = -(1/m)\, \Sigma_{i=1}^{m} [y^{(i)} \log(g(\boldsymbol{\theta} \bullet \mathbf{x}^{(i)})) + (1-y^{(i)}) \log(1-g(\boldsymbol{\theta} \bullet \mathbf{x}^{(i)}))]$.

Preliminary facts:

$\partial/\partial\theta_j\, (\boldsymbol{\theta} \bullet \mathbf{x}^{(i)}) = [\mathbf{x}^{(i)}]_j$

$g(z) = (1+e^{-z})^{-1}$

$\partial g(z)/\partial z = \partial/\partial z[(1+e^{-z})^{-1}] = -(1+e^{-z})^{-2}\, e^{-z}(-1) = e^{-z}(1+e^{-z})^{-2} = e^{-z}\, g^2(z)$

$\partial/\partial\theta_j\, g(z) = \partial g(z)/\partial z * \partial/\partial\theta_i(z) = e^{-z}\, g^2(z)\, \partial/\partial\theta_i(z)$

$\partial/\partial\theta_j\, g(\boldsymbol{\theta} \bullet \mathbf{x}^{(i)}) = \partial g(z)/\partial z * \partial/\partial\theta_i(\boldsymbol{\theta} \bullet \mathbf{x}^{(i)}) = e^{-z}\, g^2(z)\, \partial/\partial\theta_i(\boldsymbol{\theta} \bullet \mathbf{x}^{(i)})$

$= e^{-z}\, g^2(z)\, [\mathbf{x}^{(i)}]_j$

$\partial \log(g)/\partial g = 1/g$ // ie, log(x) = ln(x)

$\partial \log(g(z))/\partial\theta_j = \partial \log(g(z))/\partial g * \partial g(z)/\partial z * \partial z/\partial\theta_i$ // triple chain rule

$= (1/g) * e^{-z}\, g^2(z) * \partial z/\partial\theta_i = e^{-z}\, g(z)\, \partial z/\partial\theta_i$

$\partial \log(g(\boldsymbol{\theta} \bullet \mathbf{x}^{(i)}))/\partial\theta_j = e^{-z}\, g(z)\, [\mathbf{x}^{(i)}]_j$

$\partial \log(1-g)/\partial g = 1/(1-g) * (-1) = -1/(1-g)$

$\partial \log(1-g(z))/\partial\theta_j = \partial \log(1-g(z))/\partial g * \partial g(z)/\partial z * \partial z/\partial\theta_i$

$= [-1/(1-g)]\, [e^{-z}\, g^2(z)]\, \partial z/\partial\theta_i = -e^{-z}\, g^2(z)/(1-g) * \partial z/\partial\theta_i$

$\partial \log(1-g(\boldsymbol{\theta} \bullet \mathbf{x}^{(i)}))/\partial\theta_j = -e^{-z}\, g^2(z)/(1-g) * [\mathbf{x}^{(i)}]_j$

The key results here are these

$\partial \log(g(\boldsymbol{\theta} \bullet \mathbf{x}^{(i)}))/\partial\theta_j = e^{-z}\, g\, [\mathbf{x}^{(i)}]_j$

$\partial \log(1- g(\boldsymbol{\theta} \bullet \mathbf{x}^{(i)}))/\partial\theta_j = - e^{-z} g^2 /(1-g) * [\mathbf{x}^{(i)}]_j$

To simplify a bit, consider that (here are those mysterious properties of the g function that help)

$1-g = 1 - 1/(1+e^{-z}) = [(1 + e^{-z}) - 1]/ (1+e^{-z}) = e^{-z}/ (1+e^{-z}) = e^{-z}g$

$g^2 /(1-g) = g^2/(e^{-z}g) = g\, e^{z}$

Therefore we have

$\partial \log(g(\boldsymbol{\theta} \bullet \mathbf{x}^{(i)}))/\partial\theta_j = g\, e^{-z} [\mathbf{x}^{(i)}]_j$

$\partial \log(1- g(\boldsymbol{\theta} \bullet \mathbf{x}^{(i)}))/\partial\theta_j = - g\, [\mathbf{x}^{(i)}]_j$

The derivative of the cost function is therefore (where $z = \boldsymbol{\theta} \bullet \mathbf{x}^{(i)}$)

$\partial J(\theta)/\partial\theta_j =$

$\partial/\partial\theta_i \{ - (1/m) \Sigma_{i=1}^{m} [y^{(i)} \log(g(\boldsymbol{\theta} \bullet \mathbf{x}^{(i)})) + (1- y^{(i)}) \log(1- g(\boldsymbol{\theta} \bullet \mathbf{x}^{(i)}))] \}$

$= - (1/m) \Sigma_{i=1}^{m} [y^{(i)} g\, e^{-z} [\mathbf{x}^{(i)}]_j - (1- y^{(i)}) g [\mathbf{x}^{(i)}]_j]$

$= - (1/m) \Sigma_{i=1}^{m} [y^{(i)} e^{-z} - (1- y^{(i)})] g [\mathbf{x}^{(i)}]_j$

$= - (1/m) \Sigma_{i=1}^{m} [y^{(i)} e^{-z} - 1 + y^{(i)})] g [\mathbf{x}^{(i)}]_j$ (*)

Here is the Big Trick. At this point, we incorporate in the fact that $y^{(i)}$ can take only values 0 and 1. In this case, the following two expressions are equal (the right side being quite simple) :

$[y^{(i)} e^{-z} - 1 + y^{(i)})] g = [y^{(i)}- g]$ (**)

<u>Proof:</u> If $y^{(i)} = 1$ this says

LHS = $[y^{(i)} e^{-z} - 1 + y^{(i)})] g = e^{-z}g = 1-g$ as shown earlier
RHS = $1-g$

If $y^{(i)} = 0$ then

LHS = $[y^{(i)} e^{-z} - 1 + y^{(i)})] g = -g$
RHS = $-g$

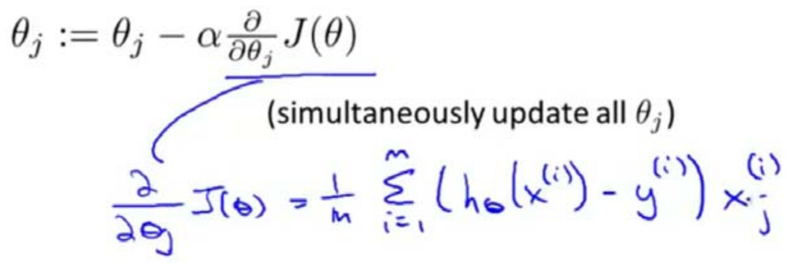
Therefore, use (**) in (*) to get

$\partial J(\theta)/\partial\theta_j = + (1/m) \Sigma_{i=1}^{m} [g - y^{(i)}] [\mathbf{x}^{(i)}]_j$

$= + (1/m) \Sigma_{i=1}^{m} [g(\boldsymbol{\theta} \bullet \mathbf{x}^{(i)}) - y^{(i)}] [\mathbf{x}^{(i)}]_j$

$= + (1/m) \Sigma_{i=1}^{m} [h_{\boldsymbol{\theta}}(x^{(i)}) - y^{(i)}] [\mathbf{x}^{(i)}]_j$

which agrees with Ng video 6-5 time code 10:14,



$$\theta_j := \theta_j - \alpha \underline{\frac{\partial}{\partial\theta_j} J(\theta)}$$

(simultaneously update all θ_j)

$$\frac{\partial}{\partial\theta_j} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_j^{(i)}$$

Notice that this is *the exact same result* as was obtained in the *analog* linear regression case, which explains why Ng added (1/2m) in that case and (1/m) in the digital case. Of course in the analog case the model fitting function was $h_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\theta} \bullet \mathbf{x}$, while in the digital case it is $h_{\boldsymbol{\theta}}(\mathbf{x}) = g(\boldsymbol{\theta} \bullet \mathbf{x})$. I think this is rather amazing, and can be credited to the g(z) function.

So this is the gradient that gets used in the logistical regression gradient descent method.

Is there a "normal equation" in this case? We have

$$(1/m) \Sigma_{i=1}^{m} [(1 + \exp(-\boldsymbol{\theta} \bullet \mathbf{x}^{(i)})^{-1} - y^{(i)}] [\mathbf{x}^{(i)}]_j = 0 \qquad j = 0,1,2...n$$

As before, let

$$X_{ik} \equiv [x^{(i)}]_k \qquad \boldsymbol{\theta} \bullet \mathbf{x}^{(i)} = \Sigma_{k=0}^{n} \theta_k [x^{(i)}]_k = \Sigma_{k=0}^{n} X_{ik} \theta_k = [X \boldsymbol{\theta}]_i$$

Then the "normal equations" read

$$\Sigma_{i=1}^{m} [(1 + \exp(-[X \boldsymbol{\theta}]_i)^{-1} - y^{(i)}] X_{ij} = 0 \qquad j = 0,1,2...n$$

This is a set of n+1 equations in the n+1 unknowns θ_i, but these equations are not linear in θ_i so linear algebra does not provide a solution as it did in the analog case. One could solve them numerically, but that is basically what gradient descent does. So we no longer have a closed form solution for **θ** in the case of logistical regression.

6. Regularization

The regularized analog cost function is taken to be

$$J(\boldsymbol{\theta}) = (1/2m) \Sigma_{i=1}^{m} [\Sigma_{j=0}^{n} \boldsymbol{\theta} \bullet \mathbf{x}^{(i)} - y^{(i)}]^2 \quad + \quad (1/2m)\lambda \, \Sigma_{j=1}^{n} \theta_j^{2}$$

so the derivatives will then be

$$\partial J(\boldsymbol{\theta})/\partial\theta_j \;= (1/m)\, \Sigma_{i=1}^{m} (h_\theta(\mathbf{x}^{(i)}) - y^{(i)})\; [x^{(i)}]_j \quad + \quad (\lambda/m)\, \theta_j\, (\, 1 - \delta_{j,0})$$

where the factor $(1 - \delta_{j,0})$ removes the λ factor when $j = 0$. The gradient descent step is then

$$\theta_j \;:=\; \theta_j - \alpha\; \{\; (1/m)\, \Sigma_{i=1}^{m} (h_\theta(\mathbf{x}^{(i)}) - y^{(i)})\; [x^{(i)}]_j \;+\; (\lambda/m)\, \theta_j\, (\, 1 - \delta_{j,0}) \;\}$$

$$:= \theta_j[\, 1 - (\alpha\lambda/m)\, (1 - \delta_{j,0})] \;- \alpha\; \{\; (1/m)\, \Sigma_{i=1}^{m} (h_\theta(\mathbf{x}^{(i)}) - y^{(i)})\; [x^{(i)}]_j\} \qquad j = 0,1..n$$

For logistical regression the regularization is done similarly,

$$J(\boldsymbol{\theta}) = -\, (1/m)\, \Sigma_{i=1}^{m}\; [y^{(i)} \log(h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})) + (1- y^{(i)}) \log(1- h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}))\,] \;+\; (1/2m)\, \lambda\, \Sigma_{j=1}^{n}\theta_j^{2}$$

and the derivatives then become

$$\partial J(\boldsymbol{\theta})/\partial\theta_j = (1/m)\, \Sigma_{i=1}^{m}\, [\, h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)}]\, [\mathbf{x}^{(i)}]_j \;+ (\lambda/m)\, \theta_j\, (\, 1 - \delta_{j,0})$$

which is the same as in the analog case in terms of $h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})$. The gradient descent step is also the same as shown above in terms of $(h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})$.

The purpose of doing regularization is to remove unwanted high frequency information from the fitting function so one avoids "overfitting". Ng comments on this in his lectures. High frequency stuff is generated when some of the θ_i get "large" (think polynomial with some large coefficients), so the regularization term increases the cost when θ_i get "large", adding a bias therefore toward smooth functions.

7. Modification of the Linear Regression method for a Vector Output

The training pairs are now $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$ where $\mathbf{x}^{(i)}$ has n+1 components and $\mathbf{y}^{(i)}$ has K components. The L^2 metric (squared) will then be this

$$d^2(\mathbf{h}_\theta(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}) \;= (\, \mathbf{h}_\theta(\mathbf{x}^{(i)}) - \mathbf{y}^{(i)})^2$$

$$= (\, \mathbf{h}_\theta(\mathbf{x}^{(i)}) - \mathbf{y}^{(i)}) \bullet (\, \mathbf{h}_\theta(\mathbf{x}^{(i)}) - \mathbf{y}^{(i)})$$

$$= \Sigma_{s=1}^{K}\, (\, [\mathbf{h}_\theta(\mathbf{x}^{(i)})]_s - [\mathbf{y}^{(i)}]_s)^2 \quad .$$

The linear *scalar* fitting function was h_θ, where $\boldsymbol{\theta}$ is a vector with components θ_j,

$$h_\theta(\mathbf{x}) = \;\; \boldsymbol{\theta} \bullet \mathbf{x} \;\; = \Sigma_{j=0}^{n}\, \theta_j x_j$$

but now we will need a linear *vector* fitting function $\mathbf{h}_\theta$,

$$\mathbf{h}_\theta(\mathbf{x}) = \theta\,\mathbf{x}$$

where now θ is a matrix with elements θ_{ij} . The sth component of the above is this:

$$[\mathbf{h}_\theta(\mathbf{x})]_s = [\theta\,\mathbf{x}]_s = \Sigma_{j=0}{}^n\theta_{sj}x_j \qquad\qquad s = 1,2....K\ .$$

The squared metric above can then be written

$$\begin{aligned} d^2(\mathbf{h}_\theta(\mathbf{x}^{(i)}), \mathbf{y}^{(i)}) &= (\ \mathbf{h}_\theta(\mathbf{x}^{(i)}) - \mathbf{y}^{(i)})^2 \\ &= (\theta\mathbf{x}^{(i)} - \mathbf{y}^{(i)})^2 \\ &= \Sigma_{s=1}{}^K\ ([\theta\mathbf{x}^{(i)}]_s - [\mathbf{y}^{(i)}]_s)^2 \\ &= \Sigma_{s=1}{}^K\ (\Sigma_{j=0}{}^n\theta_{sj}[\mathbf{x}^{(i)}]_j - [\mathbf{y}^{(i)}]_s)^2 \end{aligned}$$

Remember that the metric $d(\mathbf{h}_\theta(\mathbf{x}), \mathbf{y})$ is a measure of the distance between the value $\mathbf{y}$ and our linear fit to the value $\mathbf{y}$ which is $\mathbf{h}_\theta(\mathbf{x})$, so d is an error estimate. The analog cost function is then a sum of all these squared error estimates times a constant (1/2m),

$$\begin{aligned} J(\boldsymbol{\theta}) &= (1/2m)\ \Sigma_{i=1}{}^m\ d^2(\mathbf{h}_\theta(\mathbf{x}^{(i)}), \mathbf{y}^{(i)}) \\ &= (1/2m)\ \Sigma_{i=1}{}^m\ \{\ \Sigma_{s=1}{}^K\ (\Sigma_{j=0}{}^n\theta_{sj}[\mathbf{x}^{(i)}]_j - [\mathbf{y}^{(i)}]_s)^2\} \\ &= (1/2m)\ \Sigma_{i=1}{}^m\ \Sigma_{s=1}{}^K\ (\ \Sigma_{j=0}{}^n\theta_{sj}[\mathbf{x}^{(i)}]_j - [\mathbf{y}^{(i)}]_s\ )^2\ . \end{aligned}$$

Now that we are used to components like $[\mathbf{x}^{(i)}]_k$, we write them more simply as $x^{(i)}{}_k$ so that

$$J(\boldsymbol{\theta}) = (1/2m)\ \Sigma_{i=1}{}^m\ \Sigma_{s=1}{}^K\ (\ \Sigma_{j=0}{}^n\theta_{sj}x^{(i)}{}_j - y^{(i)}{}_s\ )^2\ . \qquad\qquad // \text{ vector cost}$$

This can then be compared to the scalar cost function

$$J(\boldsymbol{\theta}) = (1/2m)\ \Sigma_{i=1}{}^m \qquad (\Sigma_{j=0}{}^n\ \theta_j\ x^{(i)}{}_j\ -\ y^{(i)})^2\ . \qquad\qquad // \text{ scalar cost}$$

In both cases, $\mathbf{x}^{(i)}$ is a vector of dimension n+1 (n = number of real features). In the scalar case y is a scalar, but in the vector case it is a vector with K components.

The time has now come to compute the derivatives in the vector case (which of course includes the scalar case when K = 1 where $\theta_{1j} = \theta_j$).

$$J(\boldsymbol{\theta}) = (1/2m)\ \Sigma_{i=1}{}^m\ \Sigma_{s=1}{}^K\ (\ \Sigma_{j=0}{}^n\theta_{sj}x^{(i)}{}_j - y^{(i)}{}_s\ )^2$$

$$\partial J/\partial\theta_{ab} = \partial/\partial\theta_{ab}\ \{\ (1/2m)\ \Sigma_{i=1}{}^m\ \Sigma_{s=1}{}^K\ (\ \Sigma_{j=0}{}^n\theta_{sj}x^{(i)}{}_j - y^{(i)}{}_s\ )^2\ \}$$

$$= (1/2m)\ \Sigma_{i=1}{}^m\ \Sigma_{s=1}{}^K\ 2(\ \Sigma_{j=0}{}^n\theta_{sj}x^{(i)}{}_j - y^{(i)}{}_s\ )\ \partial/\partial\theta_{ab}(\ \Sigma_{j'=0}{}^n\theta_{sj'}x^{(i)}{}_{j'} - y^{(i)}{}_s\ )$$

Notice the little detail that the dummy index on the right is j' so we don't confuse it with the sum in the first factor. Continuing:

$$= (1/m)\, \Sigma_{i=1}{}^{m}\, \Sigma_{s=1}{}^{K}\, (\, \Sigma_{j=0}{}^{n}\theta_{sj}x^{(i)}{}_{j} - y^{(i)}{}_{s}\,)\, (\, \Sigma_{j'=0}{}^{n}[\partial\theta_{sj'}/\partial\theta_{ab}]\, x^{(i)}{}_{j'}\ \ )\ .$$

Now $\partial\theta_{sj'}/\partial\theta_{ab} = \delta_{s,a}\, \delta_{j',b}$ (you only get 1 if both indices match) so we continue

$$= (1/m)\, \Sigma_{i=1}{}^{m}\, \Sigma_{s=1}{}^{K}\, (\, \Sigma_{j=0}{}^{n}\theta_{sj}x^{(i)}{}_{j} - y^{(i)}{}_{s}\,)(\, \Sigma_{j'=0}{}^{n}[\delta_{s,a}\, \delta_{j',b}]\, x^{(i)}{}_{j'}\ \ )\ .$$

The $\delta_{s,a}$ kills off the $\Sigma_{s=1}{}^{K}$ and sets any occurrence of s to a,

$$= (1/m)\, \Sigma_{i=1}{}^{m}\, (\, \Sigma_{j=0}{}^{n}\theta_{aj}x^{(i)}{}_{j} - y^{(i)}{}_{a}\,)\, (\, \Sigma_{j'=0}{}^{n}[\delta_{j',b}]\, x^{(i)}{}_{j'}\ \ )\ .$$

The $\delta_{j',b}$ kills off the $\Sigma_{j'=0}$ and sets any occurrence of j' to b, so

$$= (1/m)\, \Sigma_{i=1}{}^{m}\, (\, \Sigma_{j=0}{}^{n}\theta_{aj}x^{(i)}{}_{j} - y^{(i)}{}_{a}\,)\, (\, x^{(i)}{}_{b}\ \ )\ .$$

Therefore we have shown that

$$\partial J(\theta)/\partial\theta_{ab} = (1/m)\, \Sigma_{i=1}{}^{m}\, (\, \Sigma_{j=0}{}^{n}\theta_{aj}x^{(i)}{}_{j} - y^{(i)}{}_{a}\,)\, x^{(i)}{}_{b}$$

$$= \ (1/m)\, \Sigma_{i=1}{}^{m}\, (\, [\theta\mathbf{x}^{(i)}]_{a} - y^{(i)}{}_{a}\,)\, x^{(i)}{}_{b}$$

$$= \ (1/m)\, \Sigma_{i=1}{}^{m}\, (\mathbf{h}_{\theta}(\mathbf{x}^{(i)})_{a} - y^{(i)}{}_{a}\,)\, x^{(i)}{}_{b} \qquad a = 1,2...K \quad b = 0,1...n$$

We can compare the vector derivative with the scalar one:

$$\partial J(\theta)/\partial\theta_{ab} = (1/m)\, \Sigma_{i=1}{}^{m}\, (\mathbf{h}_{\theta}(\mathbf{x}^{(i)})_{a} - \mathbf{y}^{(i)}{}_{a}\,)\, \mathbf{x}^{(i)}{}_{b} \qquad a = 1,2...K \quad b = 0,1...n \ \ \text{// vector}$$

$$\partial J(\boldsymbol{\theta})/\partial\theta_{b} = (1/m)\, \Sigma_{i=1}{}^{m}\, (h_{\theta}(\mathbf{x}^{(i)}) - y^{(i)})\, \mathbf{x}^{(i)}{}_{b} \quad . \qquad b = 0,1...n \ \ \text{// scalar}$$

For regularization, we would add $(1/2m)\, \lambda\, \Sigma_{s=1}{}^{K}\ \Sigma_{j=1}{}^{n}\, \theta_{sj}{}^{2}$ to $J(\theta)$ and then

$$\partial/\partial\theta_{ab}\, [(1/2m)\, \lambda\, \Sigma_{s=1}{}^{K}\ \Sigma_{j=1}{}^{n}\, \theta_{sj}{}^{2}]$$

$$= (1/2m)\, \lambda\, \Sigma_{s=1}{}^{K}\ \Sigma_{j=1}{}^{n}\, 2\, \theta_{sj}\, (\partial\theta_{sj}/\partial\theta_{ab})$$

$$= (1/2m)\, \lambda\, \Sigma_{s=1}{}^{K}\ \Sigma_{j=1}{}^{n}\, 2\, \theta_{sj}\ \ (\delta_{s,a}\delta_{j,b})$$

$$= (1/2m)\, \lambda\, \Sigma_{j=1}{}^{n}\, 2\, \theta_{aj}\ \ (\delta_{j,b})$$

The Σ_{j} gets no hit if b = 0, so we express this fact using the $(1-\delta_{b,0})$ factor,

$$\Sigma_{j=1}{}^{n}\, 2\, \theta_{aj}\ (\delta_{j,b}) \ =\ 2\theta_{ab}(1-\delta_{b,0})$$

and then continuing the above

$$= (1/2m)\, \lambda\, 2\theta_{ab}(1-\delta_{b,0})$$

$$=\ (\lambda/m)\, \theta_{ab}(1-\delta_{b,0})$$

and this is then the adder to the derivative. To summarize the regularized vector result :

$$J(\boldsymbol{\theta}) =\ (1/2m)\, \Sigma_{i=1}{}^{m}\, \Sigma_{s=1}{}^{K} (\ \Sigma_{j=0}{}^{n}\theta_{sj}x^{(i)}{}_{j} - y^{(i)}{}_{s}\ )^2\ +\ (1/2m)\, \lambda\, \Sigma_{s=1}{}^{K}\ \Sigma_{j=1}{}^{n}\, \theta_{sj}{}^2$$

$$=\ (1/2m)\, \Sigma_{i=1}{}^{m}\, \Sigma_{s=1}{}^{K} (\ [\theta\mathbf{x}^{(i)}]_s - y^{(i)}{}_{s}\ )^2\ +\ (1/2m)\, \lambda\, \Sigma_{s=1}{}^{K}\ \Sigma_{j=1}{}^{n}\, \theta_{sj}{}^2$$

$$=\ (1/2m)\, \Sigma_{i=1}{}^{m}\, \Sigma_{s=1}{}^{K} (h_\theta(\mathbf{x}^{(i)})_s - y^{(i)}{}_{s}\ )^2\ +\ (1/2m)\, \lambda\, \Sigma_{s=1}{}^{K}\ \Sigma_{j=1}{}^{n}\, \theta_{sj}{}^2$$

$$=\ (1/2m)\, \Sigma_{i=1}{}^{m}\ (\mathbf{h}_\theta(\mathbf{x}^{(i)}) - \mathbf{y}^{(i)})^2\ +\ (1/2m)\, \lambda\, \Sigma_{s=1}{}^{K}\ \Sigma_{j=1}{}^{n}\, \theta_{sj}{}^2$$

$$\partial J(\theta)/\partial\theta_{ab}\ = (1/m)\, \Sigma_{i=1}{}^{m}\, (\mathbf{h}_\theta(\mathbf{x}^{(i)})_a - \mathbf{y}^{(i)}{}_{a}\ )\ \mathbf{x}^{(i)}{}_{b}\ + (\lambda/m)\, \theta_{ab}(1-\delta_{b,0})$$

$a = 1,2...K$ $b = 0,1...n$ // vector

and as usual we want to compare this to the scalar regularized result from above,

$$\partial J(\boldsymbol{\theta})/\partial\theta_b\ = (1/m)\, \Sigma_{i=1}{}^{m}\, (h_\theta(\mathbf{x}^{(i)}) - \mathbf{y}^{(i)})\ \ \mathbf{x}^{(i)}{}_{b}\ +\ (\lambda/m)\, \theta_b\, (1-\delta_{b,0}) \qquad \text{// scalar}$$

Is there a **normal equation** for the un-regularized vector problem?

$$\partial J(\theta)/\partial\theta_{ab}\ = (1/m)\, \Sigma_{i=1}{}^{m}\, (\mathbf{h}_\theta(\mathbf{x}^{(i)})_a - y^{(i)}{}_{a}\ )\ x^{(i)}{}_{b}\ = 0 \qquad a = 1,2...K \quad b = 0,1...n$$

or

$$\Sigma_{i=1}{}^{m}\, (\mathbf{h}_\theta(\mathbf{x}^{(i)})_a - y^{(i)}{}_{a}\ )\ x^{(i)}{}_{b}\ = 0$$

$$\Sigma_{i=1}{}^{m}\, ([\theta\ \mathbf{x}^{(i)}]_a - y^{(i)}{}_{a}\ )\ x^{(i)}{}_{b}\ = 0$$

$$\Sigma_{i=1}{}^{m}\, (\ \Sigma_{j=0}{}^{n}\ \theta_{aj}\, x^{(i)}{}_{j} - y^{(i)}{}_{a}\ )\ x^{(i)}{}_{b}\ = 0$$

so that

$$\Sigma_{i=1}{}^{m}\, (\ \Sigma_{j=0}{}^{n}\ \theta_{aj}\, x^{(i)}{}_{j})\ x^{(i)}{}_{b}\ \ =\ \Sigma_{i=1}{}^{m}\, y^{(i)}{}_{a}\, x^{(i)}{}_{b}$$

As for the scalar normal equation, define

$$X_{ik}\ \equiv\ [x^{(i)}]_k$$

and now also

$$Y_{ik} \equiv [y^{(i)}]_k.$$

Then the above equation is

$$\Sigma_{i=1}^{m} \Sigma_{j=0}^{n} \theta_{aj} X_{ij} X_{ib} = \Sigma_{i=1}^{m} Y_{ia} X_{ib}$$

$$\Sigma_{i=1}^{m} \Sigma_{j=0}^{n} X^{T}_{bi} X_{ij} \theta^{T}_{ja} = \Sigma_{i=1}^{m} X^{T}_{bi} Y_{ia}$$

which can be expressed as the ba component of a matrix equation,

$$[X^T X \theta^T]_{ba} = [X^T Y]_{ba}$$

$$(X^T X)\theta^T = X^T Y$$

$$\theta^T = (X^T X)^{-1} X^T Y \qquad \text{// vector}$$

and this then is the exact solution to the vector linear regression problem. For example

$$\theta_{ba} = \theta^{T}_{ab} = [\ (X^T X)^{-1} X^T Y\ ]_{ab}$$

We can as usual compare this to the scalar normal equation found earlier,

$$\boldsymbol{\theta} = (X^T X)^{-1} X^T \mathbf{y} \qquad \text{// scalar}$$

8. Modification of the Logistical Regression method for a Vector Output of Bits

Start with the scalar cost function from section 4 above,

$$J(\boldsymbol{\theta}) = (1/m)\, \Sigma_{i=1}^{m} \,[-y^{(i)} \log(h_{\theta}(\mathbf{x}^{(i)})) - (1- y^{(i)}) \log(1- h_{\theta}(\mathbf{x}^{(i)}))\ ]$$

This is based on a single bit result which is 0 or 1 and uses this metric

$$d(h_{\theta}(\mathbf{x}),y) = -y \log(h_{\theta}(\mathbf{x})) - (1-y) \log(1- h_{\theta}(\mathbf{x})).$$

Consider a multi-bit output y_k where k labels the bits 1 to K. You might then consider the following as a metric to represent the distance between prediction $\mathbf{h}_{\theta}(\mathbf{x})$ and output $\mathbf{y}$ (ie, just sum the single bit metrics),

$$d(\mathbf{h}_{\theta}(\mathbf{x}),\mathbf{y}) = \Sigma_{k=1}^{K} d([h_{\theta}(\mathbf{x})]_k, y_k) = \Sigma_{k=1}^{K} [-y_k \log([h_{\theta}(\mathbf{x})]_k) - (1-y_k) \log(1- [h_{\theta}(\mathbf{x})]_k)]$$

and then the linear fitting functions could be taken as

$$[h_{\theta}(\mathbf{x})]_k = g(\Sigma_{j=0}^{n} \theta^{(k)}{}_j x_j) = g(\boldsymbol{\theta}^{(k)} \bullet \mathbf{x}) \ .$$

Here $\boldsymbol{\theta}^{(k)}$ is the vector solution that goes with the k^{th} bit. It is useful then to define a matrix θ whose elements are

$$\theta_{ki} \equiv [\boldsymbol{\theta}^{(k)}]_i \qquad k = 1,2..K \qquad i = 0,1...n \text{ (features index)}$$

Then

$$\boldsymbol{\theta}^{(k)} \bullet \mathbf{x} = \Sigma_{j=0}^{n}[\boldsymbol{\theta}^{(k)}]_j\, x_j = \Sigma_{j=0}^{n}\, \theta_{kj}\, x_j = [\theta\mathbf{x}]_k$$

so one can then write

$$[h_\theta(\mathbf{x})]_k = g(\Sigma_{j=0}^{n}\theta^{(k)}{}_j x_j) = g(\boldsymbol{\theta}^{(k)} \bullet \mathbf{x}) = g([\theta\mathbf{x}]_k) \ .$$

So the multi-bit output cost function is going to be,

$$J(\boldsymbol{\theta}) = (1/m)\, \Sigma_{i=1}^{m}\, d(\mathbf{h}_\theta(\mathbf{x}^{(i)}),\mathbf{y}^{(i)})$$

$$= -(1/m)\, \Sigma_{i=1}^{m}\, \Sigma_{k=1}^{K}\, [y^{(i)}{}_k \log([h_\theta(\mathbf{x}^{(i)})]_k) + (1-y^{(i)}{}_k) \log(1- [h_\theta(\mathbf{x}^{(i)})]_k)]$$

$$= -(1/m)\, \Sigma_{i=1}^{m}\, \Sigma_{k=1}^{K}\, [y^{(i)}{}_k \log([g([\theta\mathbf{x}^{(i)}]_k)) + (1-y^{(i)}{}_k) \log(1- [g([\theta\mathbf{x}^{(i)}]_k))] \ .$$

The middle line above appears in Ng video 9-1 time 3:49,

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

and one would expect to add a regularization term $(\lambda/2m)\, \Sigma_{k=1}^{K}\, \Sigma_{j=1}^{n}\, (\theta_{kj})^2$ to this cost.

Based on the single bit gradient derivative derived in Section 5 above,

$$\partial J(\theta)/\partial\theta_j == +(1/m)\, \Sigma_{i=1}^{m}\, [\, h_\theta(x^{(i)}) - y^{(i)}]\, [\mathbf{x}^{(i)}]_j \ ,$$

we would expect (we can quickly show) the gradient descent derivative for the multi-bit output case to be

$$\partial J(\theta)/\partial\theta_{kj} = J(\theta)/\partial\theta^{(k)}{}_j = +(1/m)\, \Sigma_{i=1}^{m}\, [\, \mathbf{h}_\theta(x^{(i)})_k - \mathbf{y}^{(i)}{}_k]\, [\mathbf{x}^{(i)}]_j$$

with a regularization adder of $(\lambda/m)\theta_{kj}\, (1 - \delta_{j,0})$.

The results of this section are used in the next set of notes (notes 3) regarding the backprop algorithm used for neural nets.

Notes #3 on Ng machine learning

PhL 5.18.12

Apologies: In this and later sections I use the word "bit" sometimes to mean a binary number which is 0 or 1, and sometimes to mean a "soft bit" which is real number in the range (0,1) which you could round up or down to get the best estimate of it as a hard bit. In a neural network, one would presumably round the final outputs to actual bits (as in a digital demodulator, say), but the internal "bit lines" of the net carry soft bits which are later called a_k, the node activations.

1. The Cost Function for a Neural Network

In Section 8 of the previous set of notes, the metric d and cost function J for a digital K-bit output linear fitting method based on input feature vectors $\mathbf{x}^{(i)}$ were given by (the outputs are $\mathbf{y}^{(i)}$)

$$d(\mathbf{h}_\theta(\mathbf{x}),\mathbf{y}) = \Sigma_{k=1}^{K} d([h_\theta(\mathbf{x})]_k, y_k) = \Sigma_{k=1}^{K} [-y_k \log([h_\theta(\mathbf{x})]_k) - (1-y_k) \log(1- [h_\theta(\mathbf{x})]_k)]$$

$$J(\boldsymbol{\theta}) = (1/m)\, \Sigma_{i=1}^{m} d(\mathbf{h}_\theta(\mathbf{x}^{(i)}),\mathbf{y}^{(i)})$$

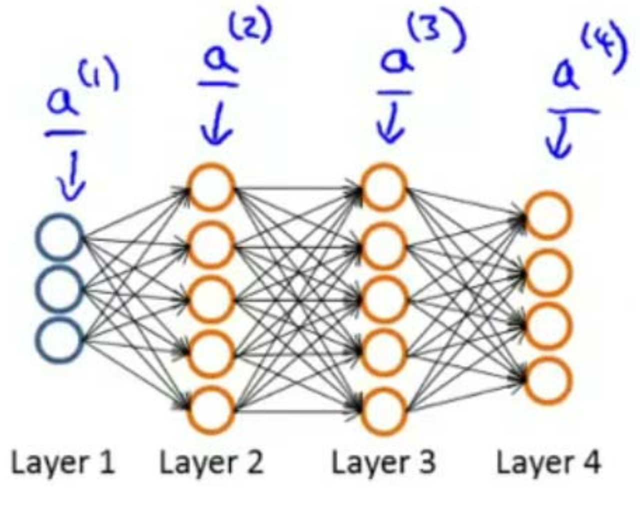
$$= -(1/m)\, \Sigma_{i=1}^{m} \Sigma_{k=1}^{K} [y^{(i)}{}_k \log([h_\theta(\mathbf{x}^{(i)})]_k) + (1-y^{(i)}{}_k) \log(1- [h_\theta(\mathbf{x}^{(i)})]_k)]$$

$\theta_{ki} \equiv [\boldsymbol{\theta}^{(k)}]_i$ = "weights" $\quad$ k = 1,2..K $\quad$ i = 0,1...n (features index)

$\boldsymbol{\theta}^{(k)} \bullet \mathbf{x} = \Sigma_{j=0}^{n}[\boldsymbol{\theta}^{(k)}]_j\, x_j = \Sigma_{j=0}^{n} \theta_{kj}\, x_j = [\theta\mathbf{x}]_k$ $\quad$ // matrix times vector

$[h_\theta(\mathbf{x})]_k = g(\Sigma_{j=0}^{n}\theta^{(k)}{}_j x_j) = g(\boldsymbol{\theta}^{(k)} \bullet \mathbf{x}) = g([\theta\mathbf{x}]_k)$.

Now consider Ng's picture of a particular neural network computer,



where $\mathbf{a}^{(1)} = \mathbf{x}$, the input vector at the left, and $\mathbf{a}^{(4)} = \mathbf{y}$ (modeled), the output vector at the right. Each circle is a computation element which computes a single bit output from several bit inputs from the previous layer. Looking at the final layer, the inputs to this layer are not $\mathbf{x}$, but are $\mathbf{a}^{(3)}$.

The number of bits in each layer is taken to be S(ℓ) [= s_ℓ Ng] so that S(4) = K, the number of output bits, and then S(1) = n+1 = the number of bits in $\mathbf{a}^{(1)} = \mathbf{x}$ including the dummy bias bit $x_0 = 1$.

In the single-layer case described above, we used

$\theta_{ki} \equiv [\boldsymbol{\theta}^{(k)}]_i$

where $\boldsymbol{\theta}^{(k)}$ was the vector for the k^{th} bit of the output. Since we are now going to have multiple layers, we need to fancy up this notation to handle all the θ's, so write

$\theta^{(\ell)}{}_{ki} \equiv [\boldsymbol{\theta}^{(k;\ell)}]_i$ // this is just $[\boldsymbol{\theta}^{(k)}]_i$ for layer $\boldsymbol{\ell}$.

We follow Ng's convention that matrix $\theta^{(\ell)}$ is the matrix of parameters which is associated with the Layer $\boldsymbol{\ell}$ which *sources* the signals going into a computation layer. This convention seems a little odd, since it means that the parameters used above in the orange computation circles of Layer 2 are called $\theta^{(1)}$. On the other hand, these $\theta^{(1)}$ are the parameters used in the "first" calculation stage above. Fine. So for the example above, we will have $\boldsymbol{\ell} = 1,2,3$.

Now what are the ranges of the lower indices on $\theta^{(\ell)}{}_{ki}$? Consider the $\boldsymbol{\ell} = 1$ case $\theta^{(1)}{}_{ki}$ for the single computation Layer1/Layer2. In this case notice that

$S(1) = S(\ell) = n + 1$ = number of inputs in vector x with bias bit added in, so $i = 0,1,2...S(\ell)$
$S(2) = S(\ell+1) = K$ = number of output bits in this case, so $k = 1,2,3.... S(\ell+1)$

Thus we enhance the line shown above to write

$\theta^{(\ell)}{}_{ki} \equiv [\boldsymbol{\theta}^{(k;\ell)}]_i$ $\quad \boldsymbol{\ell} = 1,2,3 \quad i = 0,1,2...S(\ell) \quad k = 1,2,3.... S(\ell+1)$.

and for the general network case it is the same except $\boldsymbol{\ell} = 1,2,3... L-1$.

So now we have an entire weight (parameter) matrix $\theta^{(\ell)}$ for each layer $\boldsymbol{\ell} = 1,2,3$. We include the bias bit in the i list above. If there were only 1 computation layer, it would be Layer1→Layer2 with parameters $\theta^{(1)}{}_{ki}$.

Now, here is the single-layer metric stated just above,

$$d(\mathbf{h}_\theta(\mathbf{x}),\mathbf{y}) = \Sigma_{k=1}{}^K d([h_\theta(\mathbf{x})]_k,y_k) = \Sigma_{k=1}{}^K [-y_k \log([h_\theta(\mathbf{x})]_k) - (1-y_k) \log(1- [h_\theta(\mathbf{x})]_k)]$$

$$\theta_{ki} \equiv [\boldsymbol{\theta}^{(k)}]_i \qquad [h_\theta(\mathbf{x})]_k = g(\Sigma_{j=0}{}^n \theta^{(k)}{}_j x_j) = g(\boldsymbol{\theta}^{(k)} \bullet \mathbf{x}) = g([\theta\mathbf{x}]_k)$$

We can adapt this to apply to layer 4 of the net picture above as follows: (replace the input vector $\mathbf{x}$ by $\mathbf{a}^{(3)}$ which is the input vector for layer 4; the various $a^{(\ell)}{}_k$ are called "node activations")

$$d(\mathbf{h}_\theta(\mathbf{a}^{(3)}),\mathbf{y})$$
$$= \Sigma_{s=1}{}^{S(4)}[-y_s \log(h_\theta(\mathbf{a}^{(3)})_s) - (1-y_s) \log(1- h_\theta(\mathbf{a}^{(3)})_s)]$$

$$[h_\theta(\mathbf{a}^{(3)})]_k = g(\Sigma_{j=0}{}^n \theta^{(k;3)}{}_j\, \mathbf{a}^{(3)}{}_j) = g(\boldsymbol{\theta}^{(k;3)} \bullet \mathbf{a}^{(3)}) = g([\theta^{(3)}\, \mathbf{a}^{(3)}]_k)$$

where $\quad \theta^{(3)}{}_{ki} \equiv [\boldsymbol{\theta}^{(k;3)}]_i$.

The cost function is obtained by replacing $\mathbf{y} \rightarrow \mathbf{y}^{(i)}$ and $\mathbf{x} \rightarrow \mathbf{x}^{(i)}$ and then summing the above metric on i. However, **x** is "buried" in the above line, so we need to expose it using the way the network computes things:

$$a^{(4)}{}_k = g([\theta^{(3)}\ \mathbf{a}^{(3)}]_k)$$
$$a^{(3)}{}_k = g([\theta^{(2)}\ \mathbf{a}^{(2)}]_k)$$
$$a^{(2)}{}_k = g([\theta^{(1)}\ \mathbf{a}^{(1)}]_k) = g([\theta^{(1)}\ \mathbf{x}]_k)$$

Notice that the (3) superscripts on θ and **a** match on each line, this was Ng's intention.

Now with the *careful* understanding that we are doing things "bitwise", we *could* write the above lines in vector notation as

$$\mathbf{a}^{(4)} = g(\theta^{(3)}\ \mathbf{a}^{(3)})$$
$$\mathbf{a}^{(3)} = g(\theta^{(2)}\ \mathbf{a}^{(2)})$$
$$\mathbf{a}^{(2)} = g(\theta^{(1)}\ \mathbf{a}^{(1)}) = g(\theta^{(2)}\ \mathbf{x})$$

The last two lines could be combined to give

$$\mathbf{a}^{(3)} = g(\theta^{(2)}\ g(\theta^{(2)}\ \mathbf{x}))$$
$$a^{(3)}{}_s = g(\theta^{(2)}\ g(\theta^{(2)}\ \mathbf{x})_s)$$

and then we have "exposed" the dependence of $\mathbf{a}^{(3)}$ on **x**.

This seems a very dangerous notation to me, and I would rather write things out in detail. I will use the **"Einstein convention"** of having implied sums over "repeated indices" to keep the notation from getting super messy, but major sums will still be shown explicitly. Thus for example

$$[\theta^{(2)}\ \mathbf{a}^{(2)}]_k = \theta^{(2)}{}_{k\alpha}\, a^{(2)}{}_\alpha \qquad \text{// shorthand for } \Sigma_{\alpha=0}{}^{S(2)}\ \theta^{(2)}{}_{k\alpha}\, a^{(2)}{}_\alpha\ .$$

Let's rewrite the lines from above in this manner, first changing the k indices to new names: (symbol a is now overloaded)

$$a^{(4)}{}_a = g([\theta^{(3)}\ \mathbf{a}^{(3)}]_a)$$
$$a^{(3)}{}_b = g([\theta^{(2)}\ \mathbf{a}^{(2)}]_b)$$
$$a^{(2)}{}_c = g([\theta^{(1)}\ \mathbf{a}^{(1)}]_c) = g([\theta^{(1)}\ \mathbf{x}]_c)$$

$$a^{(4)}{}_a = g(\theta^{(3)}{}_{a\alpha}\, a^{(3)}{}_\alpha)$$
$$a^{(3)}{}_b = g(\theta^{(2)}{}_{b\beta}\, a^{(2)}{}_\beta)$$
$$a^{(2)}{}_c = g(\theta^{(1)}{}_{c\gamma}\, a^{(1)}{}_\gamma) = g(\theta^{(1)}{}_{c\gamma}\, x_\gamma)$$

The last two lines can be written

$$a^{(3)}{}_b = g(\theta^{(2)}{}_{b\beta}\, a^{(2)}{}_\beta)$$
$$a^{(2)}{}_\beta = g(\theta^{(1)}{}_{\beta\gamma}\, a^{(1)}{}_\gamma) = g(\theta^{(1)}{}_{\beta\gamma}\, x_\gamma)$$

which can then be combined to give

$$a^{(3)}_{b} = g(\theta^{(2)}_{b\beta}\, g(\theta^{(1)}_{\beta\gamma}\, x_{\gamma})) \qquad \text{// implied sum on } \beta \text{ and } \gamma$$

and this shows the dependence of $\mathbf{a}^{(3)}$ on $\mathbf{x}$ with no ambiguity. We can now add a superscript (i) if we are talking about the $\mathbf{a}^{(3)}$ that arises from training sample i :

$$a^{(3,i)}_{b} = g(\theta^{(2)}_{b\beta}\, g(\theta^{(1)}_{\beta\gamma}\, x^{(i)}_{\gamma}))$$

The metric then for training sample i is then

$$d(\mathbf{h}_\theta(\mathbf{a}^{(3,i)}),\mathbf{y}^{(i)}) = \Sigma_{s=1}^{S(4)}[-y^{(i)}_s \log(h_\theta(\mathbf{a}^{(3,i)})_s) - (1-y^{(i)}_s) \log(1- h_\theta(\mathbf{a}^{(3,i)})_s)]$$

where (in all this stuff, the index s will later be called k to match Ng)

$$[h_\theta(\mathbf{a}^{(3,i)})]_s = g(\Sigma_{j=0}^{n}\theta^{(s;3)}_{j}\, a^{(3,i)}_{j}) = g(\boldsymbol{\theta}^{(s;3)} \bullet \mathbf{a}^{(3,i)}) = g([\theta^{(3)}\, \mathbf{a}^{(3,i)}]_s) \;.$$

Introducing an implied sum on b,

$$[h_\theta(\mathbf{a}^{(3,i)})]_s = g(\theta^{(3)}_{sb}\, a^{(3,i)}_{b}) \qquad \text{where} \qquad a^{(3,i)}_{b} = g(\theta^{(2)}_{b\beta}\, g(\theta^{(2)}_{\beta\gamma}\, x^{(i)}_{\gamma}))$$

we get,

$$[h_\theta(\mathbf{a}^{(3,i)})]_s = g(\theta^{(3)}_{sb}\, g(\theta^{(2)}_{b\beta}\, g(\theta^{(1)}_{\beta\gamma}\, x^{(i)}_{\gamma}))) \;.$$

Finally then we have a completely stated **cost function** in terms of the above metric for our particular neural net topology shown in the figure,

$$J(\theta^{(3)},\theta^{(2)}, \theta^{(1)}) = \Sigma_{i=1}^{m}\, d(\mathbf{h}_\theta(\mathbf{a}^{(3,i)}),\mathbf{y}^{(i)})$$

$$= \Sigma_{i=1}^{m}\, \Sigma_{s=1}^{S(4)}[-y^{(i)}_s \log(h_\theta(\mathbf{a}^{(3,i)})_s) - (1-y^{(i)}_s) \log(1- h_\theta(\mathbf{a}^{(3,i)})_s)]$$

where

$$[h_\theta(\mathbf{a}^{(3,i)})]_s = g(\theta^{(3)}_{sb}\, g(\theta^{(2)}_{b\beta}\, g(\theta^{(1)}_{\beta\gamma}\, x^{(i)}_{\gamma}))) \;.$$

Notice that the counts S(3), S(2) appear as upper limits of the implied sums in b and β. There are three closing parentheses because there are three g functions concatenated (not multiplied!).

For an arbitrary neural net picture whose last layer is L, we can see that the cost function will be

$$J(\theta^{(L-1)},\theta^{(L-2)} \,.....\theta^{(1)}) = (1/m)\, \Sigma_{i=1}^{m}\, d(\mathbf{h}_\theta(\mathbf{a}^{(L-1,i)}),\mathbf{y}^{(i)}) \qquad \textbf{// general cost function}$$

$$= (-1/m)\Sigma_{i=1}^{m}\, \Sigma_{s=1}^{S(L)}[\, y^{(i)}_s \log(h_\theta(\mathbf{a}^{(L-1,i)})_s) + (1-y^{(i)}_s) \log(1- h_\theta(\mathbf{a}^{(L-1,i)})_s)]$$

where

$$[h_\theta(\mathbf{a}^{(L-1,i)})]_s = g(\theta^{(L-1)}{}_{sa}\, g(\theta^{(L-2)}{}_{ab}\, g(\theta^{(L-3)}{}_{bc} \,\ldots\ldots\ldots\, g(\theta^{(2)}{}_{wx}\, g(\theta^{(1)}{}_{xy}\, x^{(i)}{}_{yz}))) \ldots)$$

where there are now L-1 g functions appearing, so there are L-1 closing parentheses. We have just made up sequential Latin names for the implied summation indices, a,b,c....w,x,y,z.

We could certainly go ahead and compute the gradient descent **derivatives** from this cost function. To compute $\partial J/\partial\theta^{(2)}{}_{mn}$, for example, we would have to evaluate this derivative,

$$\partial/\partial\theta^{(2)}{}_{mn}\, [h_\theta(\mathbf{a}^{(L-1,i)})]_s]$$

$$= \partial/\partial\theta^{(2)}{}_{mn} \{ g(\theta^{(L-1)}{}_{sa}\, g(\theta^{(L-2)}{}_{ab}\, g(\theta^{(L-3)}{}_{bc} \,\ldots\ldots\ldots\, g(\theta^{(2)}{}_{wx}\, g(\theta^{(1)}{}_{xy}\, x^{(i)}{}_{yz}))) \ldots)\} \,.$$

This of course is going to involve a stupendous derivative chain rule through many g functions. Here is what "chain rule" means in a simple example:

$$f(x) = a[b(c(x))] \qquad => \qquad f'(x) = a'[b(c(x))]\, b'(c(x))c'(x)$$

or

$$\frac{df}{dx} = \frac{da}{db} * \frac{db}{dc} * \frac{dc}{dx}$$

This is quite difficult to write down in the above notation, which is why the iterative back-propagation method is going to soon appear. But let's try it for a simple case:

$$[h_\theta(\mathbf{a}^{(2,i)})]_s = \Sigma_{a=1}{}^{S(2)}\, \Sigma_{b=1}{}^{S(1)} g[\theta^{(2)}{}_{sa}\, g(\theta^{(1)}{}_{ab}\, x^{(i)}{}_b)]$$

$$\partial/\partial\theta^{(2)}{}_{mn}\, [h_\theta(\mathbf{a}^{(2,i)})]_s$$

$$= \partial/\partial\theta^{(2)}{}_{mn}\, \{ \Sigma_{a=1}{}^{S(2)}\, \Sigma_{b=1}{}^{S(1)} g[\theta^{(2)}{}_{sa}\, g(\theta^{(1)}{}_{ab}\, x^{(i)}{}_b)] \}$$

$$= \partial/\partial\theta^{(2)}{}_{mn}\, \{ g[\theta^{(2)}{}_{sa}\, g(\theta^{(1)}{}_{ab}\, x^{(i)}{}_b)] \} \qquad \text{// implied sums on a and b}$$

$$= g'[\theta^{(2)}{}_{sa'}\, g(\theta^{(1)}{}_{a'b'}\, x^{(i)}{}_{b'})] * \delta_{m,s}\delta_{n,a}\, g(\theta^{(1)}{}_{ab}\, x^{(i)}{}_b)$$

$$= \delta_{m,s}\, g'[\theta^{(2)}{}_{sa'}\, g(\theta^{(1)}{}_{a'b'}\, x^{(i)}{}_{b'})] * g(\theta^{(1)}{}_{nb}\, x^{(i)}{}_b) \qquad \text{// one g'}$$

$$\partial/\partial\theta^{(1)}{}_{mn}\, [h_\theta(\mathbf{a}^{(2,i)})]_s$$

$$= \partial/\partial\theta^{(2)}{}_{mn}\, \{ g[\theta^{(2)}{}_{sa}\, g(\theta^{(1)}{}_{ab}\, x^{(i)}{}_b)] \}$$

$$= g'[\theta^{(2)}{}_{sa'}\, g(\theta^{(1)}{}_{a'b'}\, x^{(i)}{}_{b'})] * \partial/\partial\theta^{(1)}{}_{mn}\, \{\theta^{(2)}{}_{sa}\, g(\theta^{(1)}{}_{ab}\, x^{(i)}{}_b)\}$$

$$= g'[\theta^{(2)}{}_{sa'}\, g(\theta^{(1)}{}_{a'b'}\, x^{(i)}{}_{b'})] * \theta^{(2)}{}_{sa}\, g'(\theta^{(1)}{}_{ab}\, x^{(i)}{}_b)\} * \partial/\partial\theta^{(1)}{}_{mn}\, \{ \theta^{(1)}{}_{a''b''}\, x^{(i)}{}_{b''} \}$$

$$= g'[\theta^{(2)}_{sa'}\ g(\theta^{(1)}_{a'b'}\ x^{(i)}_{b'})] * \theta^{(2)}_{sa}\ g'\ (\theta^{(1)}_{ab}\ x^{(i)}_{b})\} \ * \delta_{m,a}\delta_{n,b''}\ x^{(i)}_{b''}\ \}$$

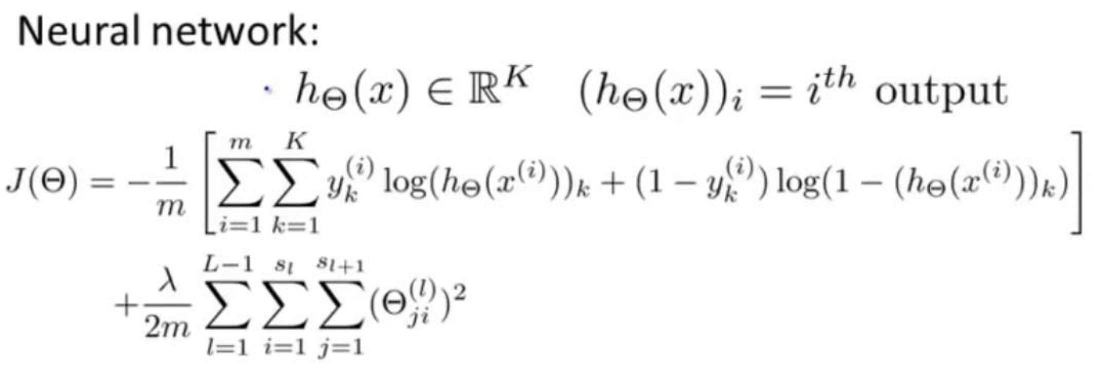
$$= g'[\theta^{(2)}_{sa'}\ g(\theta^{(1)}_{a'b'}\ x^{(i)}_{b'})] * \theta^{(2)}_{sa}\ g'\ (\theta^{(1)}_{mb}\ x^{(i)}_{b})\}\ x^{(i)}_{n}\ \} \qquad // \text{ two } g'$$

Obviously this is a messy since we have to keep writing out all the ugly g arguments. We shall try again on these derivatives in the next sections when we have compact notations for those ugly arguments.

Regularization of the above cost function means adding the usual extra terms, which in this case will be

$$J_{\text{regularization terms}} = (\lambda/2m)\ \Sigma_{\ell=1}^{L-1}\ \Sigma_{i=1}^{S(\ell)}\ \Sigma_{k=1}^{S(\ell+1)}\ [\theta^{(\ell)}_{ki}]^2$$

which compare to Ng 9-1 6:43,



Neural network:

$$h_\Theta(x) \in \mathbb{R}^K \quad (h_\Theta(x))_i = i^{th} \text{ output}$$

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1-y_k^{(i)}) \log(1-(h_\Theta(x^{(i)}))_k)\right]$$
$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2$$

where our un-regularized cost function was expressed above as (recall S(L) = K)

$$J(\theta^{(L-1)},\theta^{(L-2)}\ .....\theta^{(1)}) = \Sigma_{i=1}^{m}\ d(\mathbf{h}_\theta(\mathbf{a}^{(L-1,i)}),\mathbf{y}^{(i)}) \qquad \textbf{// general cost function}$$

$$= -(1/m)\ \Sigma_{i=1}^{m}\ \Sigma_{k=1}^{S(L)}[\ y^{(i)}_{k}\ \log(h_\theta(\mathbf{a}^{(L-1,i)})_k)\ + (1-y^{(i)}_{k})\ \log(1-\ h_\theta(\mathbf{a}^{(L-1,i)})_k)]$$

where

$$[h_\theta(\mathbf{a}^{(L-1,i)})]_k = g(\theta^{(L-1)}_{ka}\ g(\theta^{(L-2)}_{ab}\ g(\theta^{(L-3)}_{bc}\ ....\ g(\theta^{(2)}_{wx}\ g(\theta^{(1)}_{xy}\ x^{(i)}_{yz})))...)$$

Now since $\mathbf{a}^{(L-1,i)}$ is in fact a (complicated) function of $\mathbf{x}^{(i)}$ as shown, we could write

$$h_\theta(\mathbf{a}^{(L-1,i)}(\ \mathbf{x}^{(i)})) \ = \hat{h}_\theta(\mathbf{x}^{(i)})$$

where $\hat{h}_\theta$ has a different functional form of its arguments than h_θ . For example, suppose $y = x^2$ and we have

$$f(y(x)) = \hat{f}(x) = f(x^2) \qquad\qquad y = x^2$$

so that $\hat{f}(2) = f(4) \neq f(2)$, so technically one should have some marking to distinguish the functional form, since we just showed that $\hat{f}(2) \neq f(2)$. It is in this sense $\hat{h}_\theta(\mathbf{x}^{(i)})$ that Ng writes $h_\Theta(\mathbf{x}^{(i)})$ in his cost function above. Generally people don't bother with such "markings" and one just understands what is meant.

To summarize this section:

(1) The cost function for our sample neural network is found to be (changing s to k and setting S(4)=K)

$$J(\theta^{(3)},\theta^{(2)},\theta^{(1)}) = \Sigma_{i=1}^{m} \Sigma_{k=1}^{K}[-y^{(i)}{}_k \log(h_\theta(\mathbf{a}^{(3,i)})_k) - (1-y^{(i)}{}_k) \log(1- h_\theta(\mathbf{a}^{(3,i)})_k)]$$

where

$$[h_\theta(\mathbf{a}^{(3,i)})]_k = g(\theta^{(3)}{}_{ka}\, g(\theta^{(2)}{}_{ab}\, g(\theta^{(1)}{}_{bc}\, x^{(i)}{}_c))) \equiv \hat{h}_\theta(\mathbf{x}^{(i)})_k \quad .$$

In the above line there are implied sums on repeated indices a,b,c as follows

$c = 0,1,\ldots S(1)$ $\quad S(1) = n+1$ = number of input bits including the bias bit
$b = 0,1,\ldots S(2)$
$a = 0,1,\ldots S(3)$.

Ng refers to $S(\ell)$ as s_ℓ . The result for an arbitrary network is also stated above.

(2) The regularization terms to be added to J for an arbitrary network are

$$(\lambda/2m)\, \Sigma_{\ell=1}^{L-1}\, \Sigma_{i=1}^{S(\ell)}\, \Sigma_{k=1}^{S(\ell+1)}\, [\theta^{(\ell)}{}_{ki}]^2 \quad .$$

(3) We considered computing the derivatives of J for gradient descent, but it was a bit messy. That task is deferred to the next section where more notation is available to clean up the mess.

2. More notation and "forward propagation".

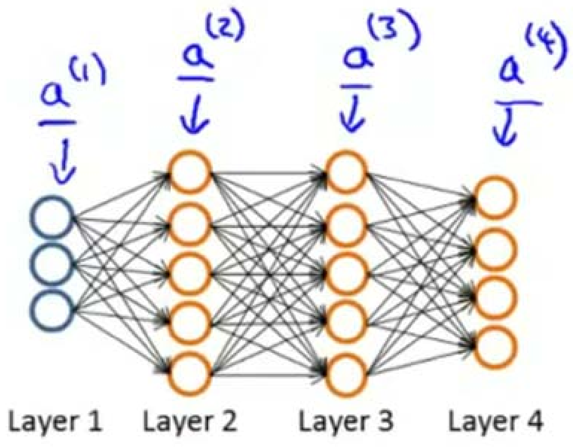
Let's go back to these results from the previous section which pertain to the Ng sample neural net studied there. The cost function is

$$J(\theta^{(3)},\theta^{(2)},\theta^{(1)}) = (1/m)\, \Sigma_{i=1}^{m}\, d(\mathbf{h}_\theta(\mathbf{a}^{(3,i)}),\mathbf{y}^{(i)})$$

$$= -(1/m)\, \Sigma_{i=1}^{m} \Sigma_{k=1}^{S(4)} [y^{(i)}{}_k \log(h_\theta(\mathbf{a}^{(3,i)})_k) + (1-y^{(i)}{}_k) \log(1- h_\theta(\mathbf{a}^{(3,i)})_k)]$$

$$\begin{aligned}[h_\theta(\mathbf{a}^{(3,i)})]_k \;=\; & a^{(4)}{}_k = g([\theta^{(3)}\, \mathbf{a}^{(3,i)}]_k) \\ & a^{(3)}{}_k = g([\theta^{(2)}\, \mathbf{a}^{(2,i)}]_k) \\ & a^{(2)}{}_k = g([\theta^{(1)}\, \mathbf{a}^{(1,i)}]_k) = g([\theta^{(1)}\, \mathbf{x}^{(i)}]_k)\end{aligned}$$

$$\mathbf{a}^{(1,i)} = \mathbf{x}^{(i)}, \qquad \mathbf{a}^{(4,i)} = \mathbf{y}^{(i)}$$



Following Ng, we define some *new symbols* to make things more compact,

$$a^{(1,i)} \equiv \mathbf{x}^{(i)}$$

$$\mathbf{z}^{(2,i)} \equiv \theta^{(1)}\, \mathbf{a}^{(1,i)} \qquad\qquad = \theta^{(1)}\, \mathbf{x}^{(i)} \qquad\qquad \mathbf{x}^{(i)} = \text{inputs}$$
$$\mathbf{a}^{(2,i)} \equiv g(\mathbf{z}^{(2,i)}) = g(\theta^{(1)}\, \mathbf{a}^{(1,i)}) \qquad = g(\theta^{(1)}\, \mathbf{x}^{(i)})$$

$$\mathbf{z}^{(3,i)} \equiv \theta^{(2)}\, \mathbf{a}^{(2,i)}$$
$$a^{(3,i)} \equiv g(\mathbf{z}^{(3,i)}) = g(\theta^{(2)}\, \mathbf{a}^{(2,i)})$$

$$\mathbf{z}^{(4,i)} \equiv \theta^{(3)}\, \mathbf{a}^{(3,i)}$$
$$a^{(4,i)} \equiv g(\mathbf{z}^{(4,i)}) = g(\theta^{(3)}\, \mathbf{a}^{(3,i)}) = \hat{h}_{\theta}(\mathbf{x}^{(i)}) \qquad = \text{model fit outputs to be compared with } y^{(i)}.$$

Ng suppresses his training sample label i, so the above appears as

$$\begin{aligned}
a^{(1)} &= x \\
z^{(2)} &= \Theta^{(1)} a^{(1)} \\
a^{(2)} &= g(z^{(2)}) \quad (\text{add } a_0^{(2)}) \\
z^{(3)} &= \Theta^{(2)} a^{(2)} \\
a^{(3)} &= g(z^{(3)}) \quad (\text{add } a_0^{(3)}) \\
z^{(4)} &= \Theta^{(3)} a^{(3)} \\
a^{(4)} &= h_\Theta(x) = g(z^{(4)})
\end{aligned}$$

where "add $a_0^{(2)}$" refers to the dummy bias bit and its parameter.

Imagine that we specify some initial starting values for all the $\theta^{(\ell)}_{ij}$ matrix parameters. Then, with that parameter set, for each training sample $\mathbf{x}^{(i)}$ we can compute the "activation values" at each node in the network following the order shown above (which is left to right or "forward") ,

$$\mathbf{a}^{(1,i)} \equiv \mathbf{x}^{(i)}$$

$\mathbf{a}^{(2,i)} = g(\theta^{(1)}\ \mathbf{a}^{(1,i)})$
$\mathbf{a}^{(3,i)} = g(\theta^{(2)}\ \mathbf{a}^{(2,i)})$
$\mathbf{a}^{(4,i)} = g(\theta^{(3)}\ \mathbf{a}^{(3,i)}) = \hat{h}_\theta(\mathbf{x}^{(i)})$

or in more detail, with implied sum on repeated index b,

$a^{(1,i)}{}_k \equiv \mathbf{x}^{(i)}{}_k$
$\mathbf{a}^{(2,i)}{}_k = g(\theta^{(1)}{}_{kb}\, a^{(1,i)}{}_b)$
$a^{(3,i)}{}_k = g(\theta^{(2)}{}_{kb}\, a^{(2,i)}{}_b)$
$a^{(4,i)}{}_k = g(\theta^{(3)}{}_{kb}\, a^{(3,i)}{}_b) = \hat{h}_\theta(\mathbf{x}^{(i)})$.

This process of computing all the activation values $a^{(\ell,i)}{}_k$ for a given training sample $\mathbf{x}^{(i)}$ is referred to as "doing forward propagation".

3. Computing the derivatives for gradient descent : preliminary calculations

Now maybe we can make more sense out of those **descent derivatives**. To this end, we will first need to compute these objects:

$$\partial/\partial\theta^{(\ell)}{}_{mn} \{ h_\theta(\mathbf{a}^{(3)})_k \} = \partial/\partial\theta^{(\ell)}{}_{mn} \{ \hat{h}_\theta(\mathbf{x})_k \}$$

Let's start with $\ell = 3$:

$$\partial/\partial\theta^{(3)}{}_{mn} \{ h_\theta(\mathbf{a}^{(3)})_k \} = \partial/\partial\theta^{(3)}{}_{mn} \{ a^{(4)}{}_k \} = \partial/\partial\theta^{(3)}{}_{mn} \{ g(z^{(4)}{}_k) \}$$

$$= g'(z^{(4)}{}_k) * \ \partial/\partial\theta^{(3)}{}_{mn} \{ z^{(4)}{}_k \} = g'(z^{(4)}{}_k) * \ \partial/\partial\theta^{(3)}{}_{mn} \{ [\theta^{(3)}\ \mathbf{a}^{(3)}]_k \}$$

$$= g'(z^{(4)}{}_k) * \ \partial/\partial\theta^{(3)}{}_{mn} \{ \theta^{(3)}{}_{k\alpha}\, a^{(3)}{}_\alpha \} \qquad \text{// implied sum on } \alpha$$

$$= g'(z^{(4)}{}_k) * \ \delta_{m,k}\delta_{n,\alpha}\, a^{(3)}{}_\alpha$$

$$= g'(z^{(4)}{}_k) * \ \delta_{m,k}\, a^{(3)}{}_n$$

Now try $\ell = 2$. We can jump down to the 3rd line of the above to get,

$$\partial/\partial\theta^{(2)}{}_{mn} \{ h_\theta(\mathbf{a}^{(3)})_k \} = g'(z^{(4)}{}_k) * \ \partial/\partial\theta^{(2)}{}_{mn} \{ \theta^{(3)}{}_{k\alpha}\, a^{(3)}{}_\alpha \}$$

$$= g'(z^{(4)}{}_k) * \ \partial/\partial\theta^{(2)}{}_{mn} \{ \theta^{(3)}{}_{k\alpha}\, g(z^{(3)}{}_\alpha) \}$$

$$= g'(z^{(4)}{}_k) * \ \theta^{(3)}{}_{k\alpha} * \ \partial/\partial\theta^{(2)}{}_{mn} \{ g(z^{(3)}{}_\alpha) \}$$

$$= g'(z^{(4)}{}_k) * \ \theta^{(3)}{}_{k\alpha} * \ g'(z^{(3)}{}_\alpha)\, \partial/\partial\theta^{(2)}{}_{mn} \{ z^{(3)}{}_\alpha \}$$

$= g'(z^{(4)}_k) * \theta^{(3)}_{k\alpha} * g'(z^{(3)}_\alpha)\, \partial/\partial\theta^{(2)}_{mn} \{ \theta^{(2)}_{\alpha\beta}\, a^{(2)}_\beta \}$ (*)

$= g'(z^{(4)}_k) * \theta^{(3)}_{k\alpha} * g'(z^{(3)}_\alpha)\, \delta_{m,\alpha}\, \delta_{n,\beta}\, a^{(2)}_\beta$

$= g'(z^{(4)}_k) * \theta^{(3)}_{km} * g'(z^{(3)}_m)\, a^{(2)}_n$

Now try $\ell = 1$ and this time we start skipping down to (*) above,

$\partial/\partial\theta^{(1)}_{mn} \{ h_\theta(\mathbf{a}^{(3)})_k \} = g'(z^{(4)}_k) * \theta^{(3)}_{k\alpha} * g'(z^{(3)}_\alpha)\, \partial/\partial\theta^{(1)}_{mn} \{ \theta^{(2)}_{\alpha\beta}\, a^{(2)}_\beta \}$

$= g'(z^{(4)}_k) * \theta^{(3)}_{k\alpha} * g'(z^{(3)}_\alpha)\, \partial/\partial\theta^{(1)}_{mn} \{ \theta^{(2)}_{\alpha\beta}\, g(z^{(2)}_\beta) \}$

$= g'(z^{(4)}_k) * \theta^{(3)}_{k\alpha} * g'(z^{(3)}_\alpha) * \theta^{(2)}_{\alpha\beta}\, \partial/\partial\theta^{(1)}_{mn} \{ g(z^{(2)}_\beta) \}$

$= g'(z^{(4)}_k) * \theta^{(3)}_{k\alpha} * g'(z^{(3)}_\alpha) * \theta^{(2)}_{\alpha\beta} * g'(z^{(2)}_\beta) \partial/\partial\theta^{(1)}_{mn} \{ z^{(2)}_\beta \}$

$= g'(z^{(4)}_k)\, \theta^{(3)}_{k\alpha}\, g'(z^{(3)}_\alpha)\, \theta^{(2)}_{\alpha\beta}\, g'(z^{(2)}_\beta) \partial/\partial\theta^{(1)}_{mn} \{ \theta^{(1)}_{\beta\gamma}\, a^{(1)}_\gamma \}$

$= g'(z^{(4)}_k)\, \theta^{(3)}_{k\alpha}\, g'(z^{(3)}_\alpha)\, \theta^{(2)}_{\alpha\beta}\, g'(z^{(2)}_\beta) \delta_{m,\beta} \delta_{n,\gamma}\, a^{(1)}_\gamma$

$= g'(z^{(4)}_k)\, \theta^{(3)}_{k\alpha}\, g'(z^{(3)}_\alpha)\, \theta^{(2)}_{\alpha m}\, g'(z^{(2)}_m)\, a^{(1)}_n$

Here is a summary of the three results above, where training index i is now restored:

$\partial/\partial\theta^{(3)}_{mn} \{ h_\theta(\mathbf{a}^{(3,i)})_k \} = [g'(z^{(4,i)}_k)\, \delta_{m,k}]\, a^{(3,i)}_n$

$\partial/\partial\theta^{(2)}_{mn} \{ h_\theta(\mathbf{a}^{(3,i)})_k \} = [g'(z^{(4,i)}_k)\, \theta^{(3)}_{km}\, g'(z^{(3,i)}_m)]\, a^{(2,i)}_n$

$\partial/\partial\theta^{(1)}_{mn} \{ h_\theta(\mathbf{a}^{(3,i)})_k \} = [g'(z^{(4,i)}_k)\, \theta^{(3)}_{k\alpha}\, g'(z^{(3,i)}_\alpha)\, \theta^{(2)}_{\alpha m}\, g'(z^{(2,i)}_m)]\, a^{(1,i)}_n$.

The pattern seems clear and it would not be hard to write these derivatives for an arbitrary net. Notice that the last line has an implied summation on repeated index α. For a larger net (larger L), as one works down the above list, there will be more such repeated indices appearing. The type of pattern seen above is typical of patterns which imply a recursion relation, and that is what the δ's are going to do below.

4. Computing the derivatives for gradient descent : completing the calculation

Armed with these required pieces of data, we can now **compute the actual descent derivatives**. First, here is the cost function from above (the total number of training samples is temporarily set to M so you see $(1/M)\Sigma_{i=1}^{M}$ out front; this is done because m is used as an index on $\theta^{(\ell)}_{mn}$. Also, we agree to overlook the overloaded fact that n is also the number of input features.)

$J(\theta^{(3)},\theta^{(2)}, \theta^{(1)}) = -(1/M)\, \Sigma_{i=1}^{M}\, \Sigma_{k=1}^{K} [y^{(i)}_k \log(h_\theta(\mathbf{a}^{(3,i)})_k) + (1-y^{(i)}_k) \log(1- h_\theta(\mathbf{a}^{(3,i)})_k)]$

Then here is the gradient derivative:

$$\partial/\partial\theta^{(\ell)}{}_{mn} [J] = -(1/M) \Sigma_{i=1}{}^{M} \Sigma_{k=1}{}^{K}[y^{(i)}{}_k\, \partial/\partial\theta^{(\ell)}{}_{mn} \log(h_\theta(\mathbf{a}^{(3,i)})_k) + (1-y^{(i)}{}_k)\, \partial/\partial\theta^{(\ell)}{}_{mn} \log(1- h_\theta(\mathbf{a}^{(3,i)})_k)]$$

$$= -(1/M) \Sigma_{i=1}{}^{M} \Sigma_{k=1}{}^{K}[y^{(i)}{}_k\, (h_\theta(\mathbf{a}^{(3,i)})_k)^{-1}\ \partial/\partial\theta^{(\ell)}{}_{mn}\{ h_\theta(\mathbf{a}^{(3,i)})_k\} + (1-y^{(i)}{}_k)\, (1- h_\theta(\mathbf{a}^{(3,i)})_k)^{-1}\ \partial/\partial\theta^{(\ell)}{}_{mn}\{(1- h_\theta(\mathbf{a}^{(3,i)})_k\}$$

$$= -(1/M) \Sigma_{i=1}{}^{M} \Sigma_{k=1}{}^{K}[y^{(i)}{}_k\, (h_\theta(\mathbf{a}^{(3,i)})_k)^{-1}\ \partial/\partial\theta^{(\ell)}{}_{mn}\{ h_\theta(\mathbf{a}^{(3,i)})_k\} - (1-y^{(i)}{}_k)\, (1- h_\theta(\mathbf{a}^{(3,i)})_k)^{-1}\ \partial/\partial\theta^{(\ell)}{}_{mn}\{(h_\theta(\mathbf{a}^{(3,i)})_k\}$$

$$= -(1/M) \Sigma_{i=1}{}^{M} \Sigma_{k=1}{}^{K}[y^{(i)}{}_k\, (h_\theta(\mathbf{a}^{(3,i)})_k)^{-1} - (1-y^{(i)}{}_k)\, (1- h_\theta(\mathbf{a}^{(3,i)})_k)^{-1}\ ]\ \partial/\partial\theta^{(\ell)}{}_{mn}\{(h_\theta(\mathbf{a}^{(3,i)})_k\}$$

$$= -(1/M) \Sigma_{i=1}{}^{M} \Sigma_{k=1}{}^{K}[y^{(i)}{}_k\, (h_\theta(\mathbf{a}^{(3,i)})_k)^{-1} - (1-y^{(i)}{}_k)\, (1- h_\theta(\mathbf{a}^{(3,i)})_k)^{-1}\ ]\ \partial/\partial\theta^{(\ell)}{}_{mn}\{(h_\theta(\mathbf{a}^{(3,i)})_k\}$$

Replace $h_\theta(\mathbf{a}^{(3,i)})_k = g(z^{(4,i)}{}_k) = g(z_k)$ for short,

$$= -(1/M) \Sigma_{i=1}{}^{M} \Sigma_{k=1}{}^{K}[y^{(i)}{}_k\, (g(z_k))^{-1} - (1-y^{(i)}{}_k)\, [(1- g(z_k))]^{-1}]\ \partial/\partial\theta^{(\ell)}{}_{mn}\{(h_\theta(\mathbf{a}^{(3,i)})_k\}$$

$$= -(1/M) \Sigma_{i=1}{}^{M} \Sigma_{k=1}{}^{K}[\ y^{(i)}{}_k\, (g(z_k))^{-1} - (1-y^{(i)}{}_k)\, [(1- g(z_k))]^{-1}\ ]\ g'(z_k) * [(1/g'(z_k)]\ \partial/\partial\theta^{(\ell)}{}_{mn}\{(h_\theta(\mathbf{a}^{(3,i)})_k\}$$

where in the last step we use the fact that all the $\partial/\partial\theta^{(\ell)}{}_{mn}\{(h_\theta(\mathbf{a}^{(3,i)})_k\}$ are proportional to $g(z_k)$. Then look at this grouping of terms which appears in the last expression above,

$$A \equiv [\ y^{(i)}{}_k\, (g(z_k))^{-1} - (1-y^{(i)}{}_k)\, (1- g(z_k))^{-1}\ ]\ g'(z_k)$$

$$= [\ y /g(z) - (1-y)/(1- g(z))\ ]\ g'(z) \qquad // \text{ abbreviating}$$

From Section 5 of the previous notes we learned that

$$g(z) = (1+e^{-z})^{-1}$$

$$g'(z) = e^{-z}\, g^2(z)$$

$$g^2(z) /(1-g(z)) = g(z)\, e^{z}$$

Therefore,

$$A = [\ y /g(z) - (1-y)/ (1- g(z))\ ]\ g'(z)$$

$$= [\ y /g(z) - (1-y)/ (1- g(z))\ ]\ e^{-z}\, g^2(z)$$

$$= [\ y\, g(z) - (1-y)g^2(z)/ (1- g(z))\ ]\ e^{-z}$$

$= [\ y\, g(z) - (1-y)\, g(z)\, e^{z}\ ]\ e^{-z}$

$= [\ y\, e^{-z} - (1-y)\ ]\ g(z)$

We also showed in that Section 5 that, since y = 0 or 1 only, this last quantity can be written as

$= (\, y - g(z)\,)$

$= [y^{(i)}_k - g(z^{(4,i)}_k)\,]$ // undoing temporary abbreviations .

Inserting this simplified version of A into our derivatives above gives

$\partial/\partial\theta^{(\ell)}_{mn}\, [J] =$

$$= -(1/M)\, \Sigma_{i=1}^{M} \Sigma_{k=1}^{K} [\ y^{(i)}_k\, (g(z_k))^{-1} - (1-y^{(i)}_k)\, [(1- g(z_k))]^{-1}\ ]\ g'(z_k)\ *$$
$$[(1/g'(z_k)]\ \partial/\partial\theta^{(\ell)}_{mn}\{(h_\theta(\mathbf{a}^{(3,i)})_k\}$$

$$= -(1/M)\, \Sigma_{i=1}^{M} \Sigma_{k=1}^{K}\, [y^{(i)}_k - g(z^{(4,i)}_k)\,]\ *\ [(1/g'(z^{(4,i)}_k)]\ \partial/\partial\theta^{(\ell)}_{mn}\{(h_\theta(\mathbf{a}^{(3,i)})_k\}$$

Now earlier we computed

$[(1/g'(z^{(4,i)}_k)]\ \partial/\partial\theta^{(3)}_{mn}\{(h_\theta(\mathbf{a}^{(3,i)})_k\}\ = a^{(3,i)}_n\, [\delta_{m,k}\,]$

$[(1/g'(z^{(4,i)}_k)]\ \partial/\partial\theta^{(2)}_{mn}\{(h_\theta(\mathbf{a}^{(3,i)})_k\}\ = a^{(2,i)}_n\, [\ \theta^{(3)}_{km}\ g'(z^{(3,i)}_m)\,]$

$[(1/g'(z^{(4,i)}_k)]\ \partial/\partial\theta^{(1)}_{mn}\{(h_\theta(\mathbf{a}^{(3,i)})_k\}\ = a^{(1,i)}_n\, [\theta^{(3)}_{k\alpha}\ g'(z^{(3,i)}_\alpha)\, \theta^{(2)}_{\alpha m}\, g'(z^{(2,i)}_m)\,]$

Installing these expressions one at a time then gives final results for our three gradient derivatives:

$\partial/\partial\theta^{(3)}_{mn}\, [J] = -(1/M)\, \Sigma_{i=1}^{M} \Sigma_{k=1}^{K}\, [y^{(i)}_k - g(z^{(4,i)}_k)\,]\ a^{(3,i)}_n\, [\delta_{m,k}\,]$

$\partial/\partial\theta^{(2)}_{mn}\, [J] = -(1/M)\, \Sigma_{i=1}^{M} \Sigma_{k=1}^{K}\, [y^{(i)}_k - g(z^{(4,i)}_k)\,]\ a^{(2,i)}_n\, [\theta^{(3)}_{km}\ g'(z^{(3,i)}_m)\,]$

$\partial/\partial\theta^{(1)}_{mn}\, [J] = -(1/M)\, \Sigma_{i=1}^{M} \Sigma_{k=1}^{K}\, [y^{(i)}_k - g(z^{(4,i)}_k)\,]\ a^{(1,i)}_n\, [\theta^{(3)}_{k\alpha}\ g'(z^{(3,i)}_\alpha)\, \theta^{(2)}_{\alpha m}\, g'(z^{(2,i)}_m)\,]$

Again, it would not be hard to generalize the above expressions to a net of arbitrary layer count L.

5. Introduction of the $\delta^{(\ell,i)}$ objects

Write the above three descent derivatives in the following form,

$\partial/\partial\theta^{(3)}_{mn}\, [J] = (1/M)\Sigma_{i=1}^{M}\, a^{(3,i)}_n\, \delta^{(4,i)}_m$

$\partial/\partial\theta^{(2)}_{mn} [J] = (1/M)\Sigma_{i=1}^{M} a^{(2,i)}_{n}\, \delta^{(3,i)}_{m}$

$\partial/\partial\theta^{(1)}_{mn} [J] = (1/M)\Sigma_{i=1}^{M} a^{(1,i)}_{n}\, \delta^{(2,i)}_{m}$

each of which is in accordance with Ng video 9-2 timecode 7:35 (Ng uses just one training sample)

$$\frac{\partial}{\partial\Theta^{(l)}_{ij}} J(\Theta) = a^{(l)}_{j}\, \delta^{(l+1)}_{i}$$

where we have now *defined* (see derivatives above!)

$\delta^{(4,i)}_{m} \equiv - \Sigma_{k=1}^{K} [y^{(i)}_{k} - g(z^{(4,i)}_{k})] [\delta_{m,k}]$

$\delta^{(3,i)}_{m} \equiv - \Sigma_{k=1}^{K} [y^{(i)}_{k} - g(z^{(4,i)}_{k})][\theta^{(3)}_{km}\ g'(z^{(3,i)}_{m})]$

$\delta^{(2,i)}_{m} \equiv - \Sigma_{k=1}^{K} [y^{(i)}_{k} - g(z^{(4,i)}_{k})] [\theta^{(3)}_{k\alpha}\ g'(z^{(3,i)}_{\alpha})\, \theta^{(2)}_{\alpha m}\, g'(z^{(2,i)}_{m})]$.

Consider :

$\theta^{(2)T}_{ma}\, \delta^{(3,i)}_{a}\, g'(z^{(2,i)}_{m})$

$= \theta^{(2)}_{am} \{ - \Sigma_{k=1}^{K} [y^{(i)}_{k} - g(z^{(4,i)}_{k})][\theta^{(3)}_{ka}\ g'(z^{(3,i)}_{a})\, g'(z^{(2,i)}_{m})]\}$

$= - \Sigma_{k=1}^{K} [y^{(i)}_{k} - g(z^{(4,i)}_{k})][\theta^{(3)}_{ka}\ g'(z^{(3,i)}_{a})\, \theta^{(2)}_{am}\, g'(z^{(2,i)}_{m})]$

$= \delta^{(2,i)}_{m}$

Next, consider :

$\theta^{(3)T}_{ma}\, \delta^{(4,i)}_{a}\, g'(z^{(3,i)}_{m})$

$= \theta^{(3)}_{am}\{-\Sigma_{i=1}^{M} \Sigma_{k=1}^{K} [y^{(i)}_{k} - g(z^{(4,i)}_{k})] [\delta_{a,k}\, g'(z^{(3,i)}_{m})]\}$

$= -(1/M)\ \Sigma_{k=1}^{K} [y^{(i)}_{k} - g(z^{(4,i)}_{k})] [\delta_{a,k}\, \theta^{(3)}_{am}\, g'(z^{(3,i)}_{m})]$

$= \delta^{(3,i)}_{m}$

Thus, we have shown these two facts explicitly

$\delta^{(3,i)}_{m} = [\theta^{(3)T}\, \delta^{(4,i)}]_{m}\, g'(z^{(3,i)}_{m})$

$\delta^{(2,i)}_{m} = [\theta^{(2)T}\, \delta^{(3,i)}]_{m}\, g'(z^{(2,i)}_{m})$.

Now look again at $\delta^{(4)}_{m}$:

$\delta^{(4,i)}{}_m \equiv - \Sigma_{k=1}{}^K [y^{(i)}{}_k - g(z^{(4,i)}{}_k)]\, [\delta_{m,k}]$

$= - [y^{(i)}{}_m - g(z^{(4,i)}{}_m)]$

$= - [y^{(i)}{}_m - a^{(4,i)}{}_m]$

$= [a^{(4,i)}{}_m - y^{(i)}{}_m]$.

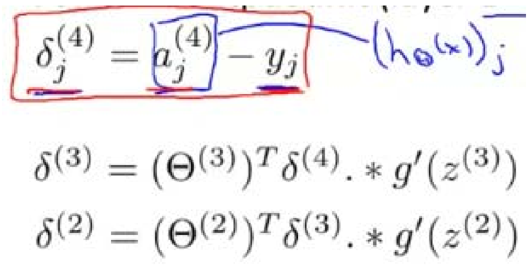
These results can be summarized as follows,

$\delta^{(4,i)}{}_m = [a^{(4,i)}{}_m - y^{(i)}{}_m] \qquad\qquad a^{(4,i)}{}_m = \hat{h}_\theta(\mathbf{x}^i)_m$

$\delta^{(3,i)}{}_m = [\theta^{(3)T}\, \delta^{(4,i)}]_m\, g'(z^{(3,i)}{}_m)$

$\delta^{(2,i)}{}_m = [\theta^{(2)T}\, \delta^{(3,i)}]_m\, g'(z^{(2,i)}{}_m)$

We can compare with Ng video 9-2 timecode 4:36 (no i label since he does only one training sample)



$$\boxed{\delta^{(4)}_j = \boxed{a^{(4)}_j} - y_j} \; (h_\Theta(x))_j$$

$$\delta^{(3)} = (\Theta^{(3)})^T\delta^{(4)} .* g'(z^{(3)})$$

$$\delta^{(2)} = (\Theta^{(2)})^T\delta^{(3)} .* g'(z^{(2)})$$

The notation .* means a component-wise multiplication which is just what we show.

The above is the "recursion" situation we have been looking for: $\delta^{(3)}$ is computed from $\delta^{(4)}$, and $\delta^{(2)}$ is computed from $\delta^{(3)}$, and so on (imagine large layer count L).

At timecode 9-2 1:44 Ng says he is working with "one training example (x,y)" so he is just trying to simplify his slides a bit. His final algorithm will have an outer loop over training samples i and he will use an accumulator variable, so he can avoid the i labels altogether.

Looking back at the algebra of g(z) we see that

$g'(z) = e^{-z}\, g^2(z) = [1/g(z) - 1]\, g^2(z) = (1-g)/g * g^2 = g(z)\,(1 - g(z))$

Thus for example

$g'(z^{(3,i)}{}_m) = g(z^{(3,i)}{}_m)\,(1 - g(z^{(3,i)}{}_m))$

But $g(z^{(3,i)}{}_m) = a^{(3,i)}{}_m$ from our notation list above, so that

$g'(z^{(3,i)}{}_m) = a^{(3,i)}{}_m\,(1 - a^{(3,i)}{}_m)$

and similarly for any other layer label. This explains Ng's blue comments at timecode 9-2 6:25m

$$\rightarrow \delta^{(3)} = (\Theta^{(3)})^T\delta^{(4)} .* \underbrace{g'(z^{(3)})}_{a^{(3)} .* (1-a^{(3)})}$$

$$\rightarrow \delta^{(2)} = (\Theta^{(2)})^T\delta^{(3)} .* \underbrace{g'(z^{(2)})}_{a^{(2)} .* (1-a^{(2)})}$$

So it is most useful to write the δ sequence in this manner:

$\delta^{(4,i)}_m = [a^{(4,i)}_m - y^{(i)}_m]$ $\qquad\qquad a^{(4,i)}_m = \hat{h}_\theta(\mathbf{x}^i)_m$

$\delta^{(3,i)}_m = [\theta^{(3)T}\,\delta^{(4,i)}]_m\, a^{(3,i)}_m\,(1 - a^{(3,i)}_m)$

$\delta^{(2,i)}_m = [\theta^{(2)T}\,\delta^{(3,i)}]_m\, a^{(2,i)}_m\,(1 - a^{(2,i)}_m)$.

6. Doing "back propagation" and the neural network gradient descent ("doing backprop")

The zeroth step is to set in some reasonable starting values for all parameters $\theta^{(\ell)}_{ij}$. As noted in video 9-6, you don't want to use all zeros. Ng suggests random initial values for $\theta^{(\ell)}_{ij}$ (random seed).

The first step is to do "forward propagation" (left to right) as described above in order to compute all the activation values $a^{(\ell,i)}_m$ for the network. These numbers are needed for each training sample i :

$a^{(1,i)}_k \equiv \mathbf{x}^{(i)}_k$
$\mathbf{a}^{(2,i)}_k = g(\theta^{(1)}_{kb}\, a^{(1,i)}_b)$
$a^{(3,i)}_k = g(\theta^{(2)}_{kb}\, a^{(2,i)}_b)$
$a^{(4,i)}_k = g(\theta^{(3)}_{kb}\, a^{(3,i)}_b) = \hat{h}_\theta(\mathbf{x}^{(i)})$.

The second step is then to compute the δ objects going in the "backward direction" (right to left),

$\delta^{(4,i)}_m = [a^{(4,i)}_m - y^{(i)}_m]$ $\qquad\qquad a^{(4,i)}_m = \hat{h}_\theta(\mathbf{x}^i)_m$
$\delta^{(3,i)}_m = [\theta^{(3)T}\,\delta^{(4,i)}]_m\, a^{(3,i)}_m\,(1 - a^{(3,i)}_m)$
$\delta^{(2,i)}_m = [\theta^{(2)T}\,\delta^{(3,i)}]_m\, a^{(2,i)}_m\,(1 - a^{(2,i)}_m)$

The third step is to compute the gradient descent derivatives as described above, now that we have all the $a^{(\ell,i)}_m$ and the $\delta^{(\ell,i)}_m$ values from the previous two steps:

$\partial/\partial\theta^{(3)}_{mn}\,[J] = (1/M)\Sigma_{i=1}^{M}\, a^{(3,i)}_n\, \delta^{(4,i)}_m$
$\partial/\partial\theta^{(2)}_{mn}\,[J] = (1/M)\Sigma_{i=1}^{M}\, a^{(2,i)}_n\, \delta^{(3,i)}_m$
$\partial/\partial\theta^{(1)}_{mn}\,[J] = (1/M)\Sigma_{i=1}^{M}\, a^{(1,i)}_n\, \delta^{(2,i)}_m$

The fourth step is to do the actual gradient descent step using these derivatives, which causes an update of all the $\theta^{(\ell)}{}_{ij}$ values and (hopefully) a resulting reduction of the cost function.

$$\theta^{(3)}{}_{mn} := \theta^{(3)}{}_{mn} - \alpha\, [\, \partial/\partial\theta^{(3)}{}_{mn}\, [J] \;\; - (\lambda/M)\, \theta^{(3)}{}_{mn}(1-\delta_{n,0})]$$
$$\theta^{(2)}{}_{mn} := \theta^{(2)}{}_{mn} - \alpha\, [\, \partial/\partial\theta^{(2)}{}_{mn}\, [J] \;\; - (\lambda/M)\, \theta^{(2)}{}_{mn}(1-\delta_{n,0})]$$
$$\theta^{(1)}{}_{mn} := \theta^{(1)}{}_{mn} - \alpha\, [\, \partial/\partial\theta^{(1)}{}_{mn}\, [J] \;\; - (\lambda/M)\, \theta^{(1)}{}_{mn}(1-\delta_{n,0})]$$

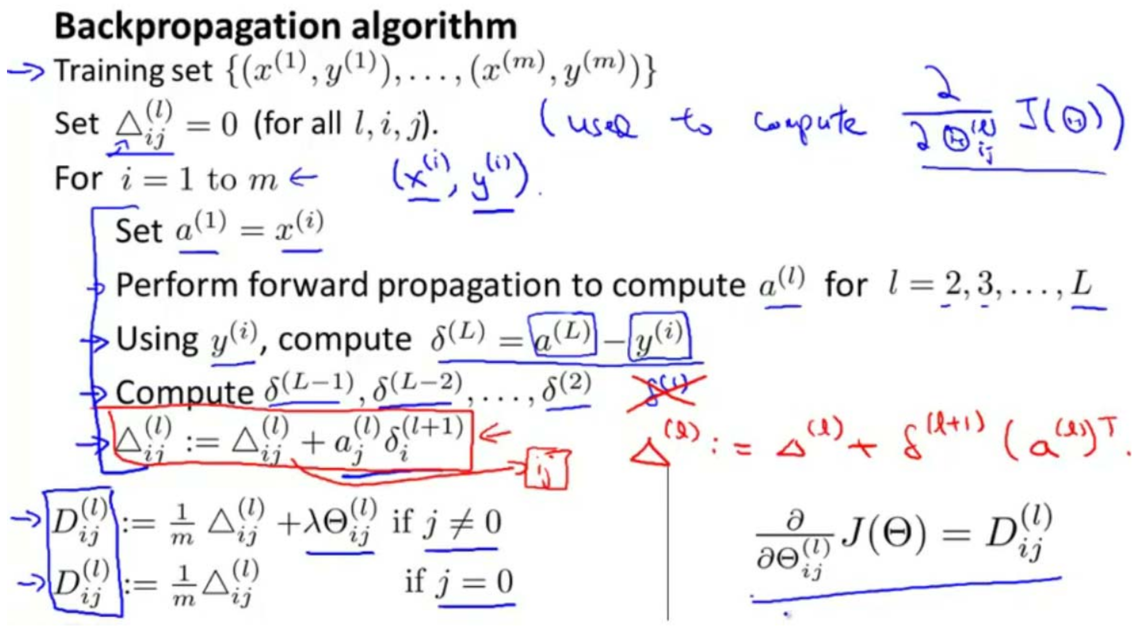
where we have added the regularization terms which match the regularization terms in the total cost function stated earlier.

The "fifth step" is to iterate the above steps 1 through 4, watching J decrease until it stabilizes at a minimum value. Hopefully one then ends up with the "best" set of weights $\theta^{(\ell)}{}_{ij}$ to fit the training data. Then you can challenge the trained system with never-been-seen input **x** and watch the **y** it generates.

In his proposed algorithm code, Ng does one training sample at a time. Accumulators $\Delta^{(\ell)}{}_{mn}$ are preset to zero, then for each training sample i, the corresponding $a^{(\ell,i)}{}_{n}\ \delta^{(\ell,i)}{}_{m}$ is computed as shown above (forward prop then backprop) and then accumulated into the accumulator. When the training samples are finished (i = 1,2...M), the accumulators hold the derivatives of interest but the 1/M factors has not been added yet, nor have the regularization values been added. This then is done in the last step which is

$$D^{(\ell)}{}_{mn} := (1/M)\Delta^{(\ell)}{}_{mn} + \lambda\, \theta^{(\ell)}{}_{mn}\ (1-\delta_{n,0})$$

so the $D^{(\ell)}{}_{mn}$ are the final gradient descent derivatives. Here from 9-2 11:21 (it all makes sense)



Backpropagation algorithm

→ Training set $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$

Set $\Delta^{(l)}_{ij} = 0$ (for all l, i, j). $\left(\text{used to compute } \frac{\partial}{\partial \Theta^{(l)}_{ij}} J(\Theta)\right)$

For $i = 1$ to m ← $(x^{(i)}, y^{(i)})$.

```
    Set $a^{(1)} = x^{(i)}$
    → Perform forward propagation to compute $a^{(l)}$ for $l = 2, 3, \ldots, L$
    → Using $y^{(i)}$, compute $\delta^{(L)} = \boxed{a^{(L)}} - \boxed{y^{(i)}}$
    → Compute $\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)}$  $\cancel{\delta^{(1)}}$
    → $\boxed{\Delta^{(l)}_{ij} := \Delta^{(l)}_{ij} + a^{(l)}_j \delta^{(l+1)}_i}$ ←   $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T.$
```

→ $\boxed{D^{(l)}_{ij}} := \frac{1}{m} \Delta^{(l)}_{ij} + \lambda \Theta^{(l)}_{ij}$ if $j \neq 0$

→ $\boxed{D^{(l)}_{ij}} := \frac{1}{m} \Delta^{(l)}_{ij}$ if $j = 0$

$$\frac{\partial}{\partial \Theta^{(l)}_{ij}} J(\Theta) = D^{(l)}_{ij}$$

7. Computation comments.

The forward prop computation is

$$a^{(3,i)}_k = g(\theta^{(2)}_{kb}\, a^{(2,i)}_b)$$

which is a small matrix times a vector (implied sum on b, "multiply-accumulate") , then an expo, add and invert (for g) and this is done for each of L-1 values of k which is a very small integer. It is a cheap calculation, hardware could kick butt on such a thing.

The backprop calculation is similarly simple,

$$\delta^{(3,i)}_m = [\theta^{(3)T}\, \delta^{(4,i)}]_m\, a^{(3,i)}_m\, (1 - a^{(3,i)}_m)$$

or

$$\delta^{(3,i)}_m = \theta^{(3)T}_{mb}\, \delta^{(4,i)}_b\ \ a^{(3,i)}_m\, (1 - a^{(3,i)}_m)$$

This is a small matrix times a vector (sum on b) then one add and two multiplies, and all this is done for each of the L-1 values of k. Very cheap, very hardware suitable.

The accumulation step involves one more multiply to get the $a^{(\ell,i)}_n\, \delta^{(\ell,i)}_m$.

It would not be hard to make an estimate of computing time for backprop for software or hardware.

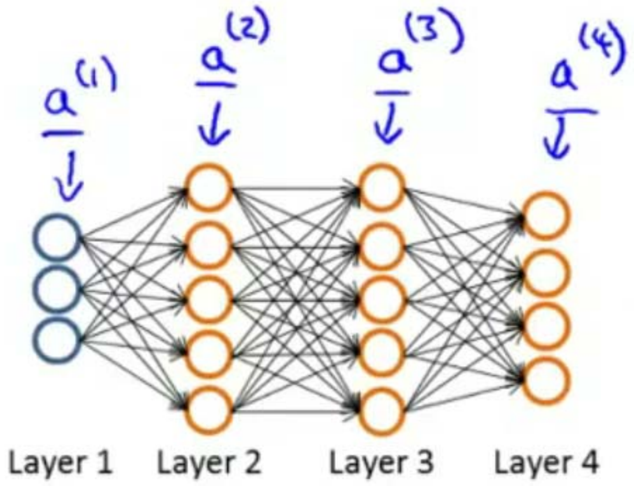
In contrast, when "gradient checking" is turned on as per video 9-5, you compute *the entire cost function* two times (+ε and -ε) for each parameter $\theta^{(\ell)}_{mn}$ and you can see that this is a long and therefore slow calculation: (notice the big sum over i, and the nesting of lots of g functions)

$$J(\theta^{(3)},\theta^{(2)}, \theta^{(1)}) = \Sigma_{i=1}{}^{m}\, \Sigma_{k=1}{}^{K}[-y^{(i)}_k \log(h_\theta(\mathbf{a}^{(3,i)})_k) - (1-y^{(i)}_k) \log(1- h_\theta(\mathbf{a}^{(3,i)})_k)]$$

$$[h_\theta(\mathbf{a}^{(3,i)})]_k = g(\theta^{(3)}_{ka}\, g(\theta^{(2)}_{ab}\, g(\theta^{(1)}_{bc}\, x^{(i)}_c))) \equiv \hat{h}_\theta(\mathbf{x}^{(i)})_k \quad .$$

8. Topology comment

The network topology considered by Ng is this



I don't know if Ng is going to mention this. One might wonder why you can't have "wires" which bypass certain layers. For example, one might want a wire going from the bottom circle of layer 2 to the bottom circle of layer 4. The topology shown above in fact allows for this possibility. One would simply add a new node below that lowest layer 3 node and have it ignore all inputs except that from the bottom of layer 2, and have it generate a non-zero output to the bottom circle of layer 4. It would have some θ parameters to cause this to happen. I would guess that in any biological net, there are such bypass wires because they would be efficient (but less "general purpose"). Perhaps there are input features that are extremely important, and they might be wired directly to the output, or directly into some middle layer.