Informal Ng Machine Learning Class Notes

Phil Lucht
Rimrock Digital Technology, Salt Lake City, Utah 84103
last update: Feb 17, 2015
rimrock@xmission.com

This file contains my informal notes related to Prof. Andrew Ng's Summer **2012** on-line Stanford/ Coursera Machine Learning class. I am just a student in the class and know only what Prof. Ng has taught in his video lectures. If you find errors and report them to me, I will update these notes.

A. Two Related Analog Regression Problems and the Normal Equation

There are two different data fitting problems that have basically the same solution.

1. Problem 1: Polynomial of Degree n Fit with One Feature.

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

$$p_n(x) = \Sigma_{j=0}^{n} \theta_j x^j = \theta_0 + \theta_1 x + \theta_2 x^2 + ... + \theta_n x^n$$

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

$$p_n(x^{(i)}) = \Sigma_{j=0}^{n} \theta_j [x^{(i)}]^j = \theta_0 + \theta_1 x^{(i)} + \theta_2 [x^{(i)}]^2 + ... + \theta_n [x^{(i)}]^n \quad .$$

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 θ_j that minimize this least squares fit cost function,

$$J(\boldsymbol{\theta}) = \Sigma_{i=1}^{m} (y^{(i)} - p_n(x^{(i)}))^2 \qquad \boldsymbol{\theta} = (\theta_0, \theta_1 .... \theta_n)$$

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

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

2. Problem 2: Linear Fit with n Features.

Take an m-element data set (training set 1,2..m) with *n features* $x^{(i)}{}_j$ (where i = 1..m labels elements of the training 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 **x**, not a polynomial]

$$p_n(\mathbf{x}) = \theta_0 + \theta_1 x_1 + \theta_1 x_2 + ... + \theta_n x_n \quad . \qquad \text{// quadratic terms would be like } \theta_{ij} x_i x_j \text{ ...}$$

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

$$p_n(\mathbf{x}^{(i)}) = \theta_0 + \theta_1 x^{(i)}{}_1 + \theta_2 x^{(i)}{}_2 + ... + \theta_n x^{(i)}{}_n \quad .$$

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

$$p_n(\mathbf{x}^{(i)}) = \theta_0 x^{(i)}{}_0 + \theta_1 x^{(i)}{}_1 + \theta_2 x^{(i)}{}_2 + ... + \theta_n x^{(i)}{}_n$$

$= \Sigma_{j=0}^{n} \theta_j \, x^{(i)}{}_j \;=\; \boldsymbol{\theta} \bullet \mathbf{x}^{(i)} \qquad \boldsymbol{\theta} = (\theta_0, \theta_1 .... \theta_n) \qquad \mathbf{x}^{(i)} = (x^{(i)}{}_0, x^{(i)}{}_1, ...)$

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

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

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

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 training point i.

3. Relation between these two problems.

Suppose you solve Problem 1 for the polynomial coefficients θ_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!

4. Solution of Problem 1: Polynomial of Degree n Fit with One Feature.

To find a minimum of J, we set all the partial derivatives with respect to the θ_i to zero. I use the notation that $\partial/\partial\theta_k$ (partial derivative with respect to θ_k) and then for short, $\partial_k \equiv \partial/\partial\theta_k$. Then

$J(\boldsymbol{\theta}) = \Sigma_{i=1}^{m} (y^{(i)} - \Sigma_{j=0}^{n} \theta_j \, [x^{(i)}]^j)^2$ // least squares fit cost function

$\partial_k \, J(\boldsymbol{\theta}) = \Sigma_{i=1}^{m} 2 \, (y^{(i)} - \Sigma_{j=0}^{n} \theta_j \, [x^{(i)}]^j) \; \partial_k \; (y^{(i)} - \Sigma_{j=0}^{n} \theta_j \, [x^{(i)}]^j)$ // chain rule

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

$= \Sigma_{i=1}^{m} 2 \, (y^{(i)} - \Sigma_{j=0}^{n} \theta_j \, [x^{(i)}]^j) \; (- \Sigma_{j=0}^{n} \; \{ \delta_{k,j} \} \, [x^{(i)}]^j)$ // $\partial\theta_j/\partial\theta_k = \delta_{k,j}$

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

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

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

or

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

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

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

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

Then the normal equations can be written as

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

or

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

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

$S\,\boldsymbol{\theta} = \mathbf{t}$ // 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)

$\boldsymbol{\theta} = S^{-1}\mathbf{t}$

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

5. Solution of Problem 2: Linear Fit with n features.

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

$\boldsymbol{\theta} = S^{-1}\mathbf{t}$ $S^{-1} = \text{cof}(S^T)/\det(S)$

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, $S_{kj} = S_{jk}$

6. The Normal Equation in Prof. Ng's form

To make this look like what Prof. 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 (i) and n+1 columns (k), not square

Then one has

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

$\Rightarrow S = X^T X$ // 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

$$\boldsymbol{\theta} = S^{-1}\mathbf{t} \ = \ (X^{T}X)^{-1} X^{T}\mathbf{y} \ .$$

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

7. What happens if two or more features measure the same thing?

In the above,

$$\det(S) = \det(X^{T}X) = \det(X^{T}) \det(X) = \det(X) \det(X) \ = [\det(X)]^{2} \ .$$

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 features are basically representations of the same feature idea. This makes two columns of X roughly the same, causing $\det(X) \approx 0$ and then $\det(S) \approx 0$ and then $S^{-1} = \text{cof}(S^{T})/\det(S)$does not exist or is very "noisy" and not meaningful.

B. Discussion of Linear Regression and Logistic Regression

1. Review of Linear Regression (ie, doing an analog data fit)

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 add a factor 1/2m in front as Prof. Ng does)

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

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

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

from which we (Section A.6) got the normal equation giving the solution value for $\boldsymbol{\theta}$,

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

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

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

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

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

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

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

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

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

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

which Prof. Ng likes to call a "**hypothesis**", but we might 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:

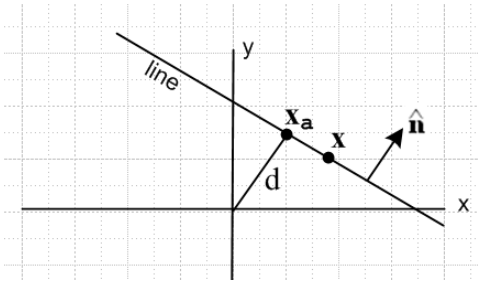
$h_{\theta}(\mathbf{x} + \mathbf{z}) = h_{\theta}(\mathbf{x}) + h_{\theta}(\mathbf{z})$ // since $\boldsymbol{\theta}\bullet (\mathbf{x} + \mathbf{z}) = \boldsymbol{\theta}\bullet\mathbf{x} + \boldsymbol{\theta}\bullet\mathbf{z}$

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

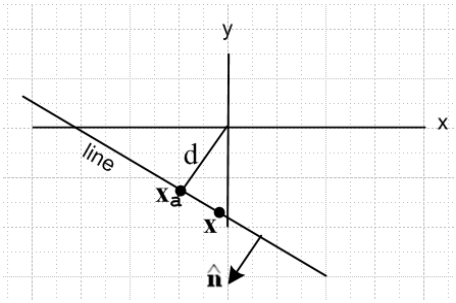
2. What is the equation of a plane?

In an the N-dimensional space $\mathcal{R}^{\mathbf{N}}$, which has points $\mathbf{x} = (x_1, x_2 ... x_N)$, the equation of a "plane" (which has dimension N-1) is given by $\mathbf{x} \bullet \hat{\mathbf{n}} = d$ where $\hat{\mathbf{n}}$ is a vector normal to the plane (pointing away from the origin) and d is the distance of closest approach of the "plane" to the origin. It is not hard to convince oneself of this fact in 2 or 3 dimensions, given the meaning of the dot product as the projection of one vector on another, and the result is valid for any number of dimensions N.

For example, consider this 2D situation,



or



The point of closest approach to the origin is called $\mathbf{x_a}$, and **x** is just some other point on the line. It seems pretty clear that vector $\mathbf{x} - \mathbf{x_a}$ (which lies along on the line) is perpendicular to $\hat{\mathbf{n}}$ so that

$(\mathbf{x} - \mathbf{x_a})\bullet \hat{\mathbf{n}} = 0$ => $\mathbf{x} \bullet \hat{\mathbf{n}} = \mathbf{x_a} \bullet \hat{\mathbf{n}}$.

Since $\mathbf{x_a}$ and **n** are aligned, we know that

$\mathbf{x_a} \bullet \hat{\mathbf{n}} = |x_a|\, |\hat{\mathbf{n}}| \cos(0) = |x_a|\, 1\ \ 1 = |x_a| = d.$

Therefore the equation of the line is simply

$\mathbf{x} \bullet \hat{\mathbf{n}} = d$

Multiplying both sides by a positive number n, and then defining $\mathbf{n} = n\,\hat{\mathbf{n}}$, one gets $\mathbf{x} \bullet \mathbf{n} = nd$ where now n is the magnitude of the normal vector **n**, which is no longer a unit vector.

Therefore, if you are handed the equation $\mathbf{x} \bullet \mathbf{n} = s$ with $s > 0$, you may conclude that the set of points **x** which solve this equation form a "plane" having normal vector **n** pointing away from the origin, and the

closest approach of this plane to the origin is d = s/n where n is the magnitude of the normal vector. If s < 0, then just write $\mathbf{x} \bullet (-\mathbf{n}) = -s = |s|$ and this is a plane with d = |s|/n but **n** points toward the origin.

Usually we don't care which of the two possible normals ±**n** we use -- both are normal to the plane. So we can summarize and just say that for the equation $\mathbf{x} \bullet \mathbf{n} = s$, **n** is normal to the plane and the distance of closest approach to the origin is d = |s|/n.

3. The best fit function for linear regression: it is really a plane.

Now consider again the best fit function $y = h_{\theta}(\mathbf{x}) = \boldsymbol{\theta} \bullet \mathbf{x}$. Given some new feature **x**, this equation gives the "best guess" of y.

If there is only one real feature x_1 (recall $x_0 = 1$) then this has the form $y(x_1) = \theta_0 + \theta_1 x_1$, or replacing the argument with x, $y = \theta_0 + \theta_1 x$. This is of course the equation of a straight line (a "plane" when N=2). Writing it in the form $-\theta_1 x + y = \theta_0$, we conclude from the above discussion that this line has normal $\mathbf{n} = (-\theta_1, 1)$ and its closest approach to the origin is $\theta_0/n = \theta_0/\sqrt{1+\theta_1^2}$.

If there are two real features x_1 and x_2, then the best fit function is $y(x_1,x_2) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$. If one regards y as the third height dimension above the plane with coordinates x_1 and x_2, then the equation $y = \theta_0 + \theta_1 x_1 + \theta_2 x_2$ is the equation of a standard issue plane in the 3D space (x_1, x_2, y). Writing this equation in the form $-\theta_1 x_1 - \theta_2 x_2 + y = \theta_0$, we conclude that this plane has normal vector $\mathbf{n} = (-\theta_1, -\theta_2, 1)$ and that its closest approach to the origin is $d = \theta_0/n = \theta_0/\sqrt{1+\theta_1^2+\theta_2^2}$.

If there are n real features $x_1 \ldots x_n$, then $y = \boldsymbol{\theta} \bullet \mathbf{x} = \theta_0 + \theta_1 x_1 + \ldots + \theta_n x_n$ is the equation of a "plane" in n+1 dimensional space. This plane has a normal vector $\mathbf{n} = (-\theta_1, -\theta_2, \ldots -\theta_n, 1)$ and its closest approach to the origin is d/n where $n = \sqrt{1+\theta_1^2+\theta_2^2 + \ldots + \theta_n^2}$. Sometimes a plane in spaces of more than 3 dimensions is called a hyperplane.

The main point of this comment is that the "best fit function" $y = h_{\theta}(\mathbf{x}) = \boldsymbol{\theta} \bullet \mathbf{x}$ is always a "plane" in a space of dimension n+1 where n is the number of real features. The "feature space" has dimension n, since $\mathbf{x} = (x_1, x_2 \ldots x_n)$. We can regard the features **x** as lying in a "plane" of n dimensions, and then the "height" $y = \boldsymbol{\theta} \bullet \mathbf{x}$ is plotted over this plane.

In particular, the best fit function for linear regression is never some kind of "curve" or "curved surface".

4. Norms, Metrics and Dot Products

The vector space (aka a linear space) R^n has elements like **x** and **y** which can be added and subtracted and stretched,

$\mathbf{x} + \mathbf{y}$ $\quad$ $\mathbf{x} - \mathbf{y}$ $\quad$ $\alpha\mathbf{x}$ $\quad$ (α = real) $\quad$ $\mathbf{x} = (x_1, x_2, \ldots x_n)$.

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 $\mathcal{R}^n$ also has the notion of the "length of a vector" || **x** || which is this:

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

where || **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 $\mathbf{L_2}$ **norm** since power 2 appears in $\Sigma_i |x_i|^2$. Thus, $\mathcal{R}^n$ is not just a vector space, it is also a normed space. Using this norm, one can then define a notion of "the distance d(**x**,**y**) between two vectors" as follows

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

and any such distance function d(**x**,**y**) is called a **metric**. With this metric, our space is a vector space, a normed space, and a metric space (this combination is sometimes called a Banach 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 $\mathcal{R}^n$ finally has another property called the **inner product** or **scalar product** or **dot product**. It is defined as ($\mathcal{R}$ means real numbers, not complex numbers)

$$\mathbf{x} \bullet \mathbf{y} \equiv \Sigma_i x_i y_i \qquad \text{// sometimes written as } (\mathbf{x},\mathbf{y}) \text{ or } \langle \mathbf{x},\mathbf{y} \rangle \text{ or } \langle \mathbf{x} \mid \mathbf{y} \rangle \text{ or } \mathbf{x}^T\mathbf{y} \text{ or } \mathbf{y}^T\mathbf{x}$$

Thus we see that

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

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

A vector space "endowed" with a norm, a metric, *and* a dot product is called a **Hilbert Space** and $\mathcal{R}^n$ is a Hilbert Space.

The dot product can be written in **matrix notation** as follows, where **y** is a column vector and **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 **x** and **y**. Thus, vectors which are perpendicular have a zero dot product since cos(π/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≠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 **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

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

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

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

5. Another review of the Ng analog linear case

The cost function from above may now be written in terms of the metric notation (omitting the 1/2m)

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

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. Prof. Ng could have used the L^1 norm and written a different cost function

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

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

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

6. Linear logistic regression (binary classification problem)

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

d(s,y) = -log(s) if y = 1
d(s,y) = -log(1-s) if y = 0 .

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

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 (both 1 or both 0), 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 $0 \le s \le 1$. Sometimes a variable like y is called a hard bit, while a variable like s is called a soft bit which can be rounded to yield a best estimate of a hard bit.

Here are some examples of metric values:

d(.99,1) = -log(.99) = .004 a very small cost or distance
d(.01,1) = -log(.01) = +2 a large cost or distance

d(.01,0) = -log(.99) = .004 a very small cost or distance
d(.99,0) = -log(.01) = +2 a large cost or distance

For $0 \le s \le 1$ 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

$d(s,y) = -y \log(s) - (1-y) \log(1-s)$ // valid for y = 1 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

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

So $\mathbf{x}$ is the vector of features in the model and $h_\theta(\mathbf{x})$ is some "model function" (hypothesis) which one hopes is useful to predict how new points $\mathbf{x}$ should be classified, based on what was learned from the training points. Prof. Ng uses the following mapping function

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

Again, 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 (0,1) and thus meets the requirements noted above for the specialized metric d(g,y) where g must lie in (0,1).

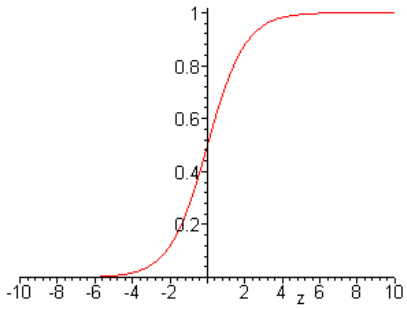
So in the analog regression model the linear model function was taken as $h_\theta(\mathbf{x}) = \boldsymbol{\theta} \bullet \mathbf{x}$, whereas in the classification regression model we shall use $h_\theta(\mathbf{x}) = g(\boldsymbol{\theta} \bullet \mathbf{x})$, where the extra function g(z) is used to get $h_\theta(\mathbf{x})$ into the required metric range (0,1) for being a soft bit.

One could imagine other functions g(z), such as $(1+e^{-5z})^{-1}$ or $(1/\pi)\tan^{-1}(z) + 1/2$, 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 classification gradient.

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

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

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

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

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

Prof. Ng has added the positive constant factor (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.

Prof. Ng. tends to favor the notation $\boldsymbol{\theta} \bullet \mathbf{x}^{(i)} = \boldsymbol{\theta}^{\mathbf{T}}\ \mathbf{x}^{(i)}$ or just $\theta^{\mathbf{T}}\ x^{(i)}$ with no bolding of the vectors. The dot product seems more useful when one is thinking in terms of geometry.

7. Computation of derivatives for logistic 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 $z = \boldsymbol{\theta} \bullet \mathbf{x}^{(i)}$)

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

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

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

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

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

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) :

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

Proof: 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

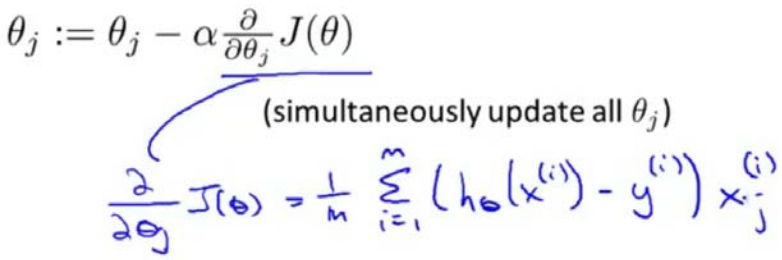
Therefore, use (**) in (*) to get

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

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

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

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



Notice that this is *the exact same result* as was obtained in the *analog* linear regression case, which explains why Prof. Ng added (1/2m) in that case and (1/m) in the classification case. Of course in the analog case the model fitting function was $h_{\theta}(\mathbf{x}) = \boldsymbol{\theta} \bullet \mathbf{x}$, while in the classification case it is $h_{\theta}(\mathbf{x}) = g(\boldsymbol{\theta} \bullet \mathbf{x})$. I think this is rather amazing, and can be credited to the g(z) function. In the analog case, $y = \boldsymbol{\theta} \bullet \mathbf{x}$ is the plane used to fit the training data, whereas in the logistic case $\boldsymbol{\theta} \bullet \mathbf{x} = 0$ is a different plane used to classify data points (see next subsection).

To summarize, we have found the gradient that gets used in the logistic regression gradient descent method, and it is this:

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

Is there a "normal equation" in this case? Setting the derivatives to zero we have

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

As before, let

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

Then the "normal equations" read

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

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 these equations numerically, but that is basically what gradient descent does. So we no longer have a closed form solution for $\boldsymbol{\theta}$ in the case of logistic regression.

8. Comment on the "boundary" in logistic regression.

In linear regression, we obtained a certain "plane" $y=\boldsymbol{\theta}\bullet\mathbf{x}$ as the best fit to the training data (see Section B.3 above). This "plane" exists in n+1 dimensional space when there are n real features.

In the binary classification "fit" which is logistic regression, we come up with a slightly different "plane". This plane exists in n dimensional space (not n+1) and has the equation $\boldsymbol{\theta}\bullet\mathbf{x}=0$. We can write this out as $\theta_0+\theta_1x_1+...+\theta_nx_n=0$, or rearranging, $-\theta_1x_1-\theta_2x_2...-\theta_nx_n=\theta_0$. Based on our discussion about planes in Section B.2 above, we see that this plane has normal $\mathbf{n}=(-\theta_1,-\theta_2,....-\theta_n)$ and the closest approach of this plane to the origin $|\theta_0|/n$ where $n=\sqrt{\theta_1^2+\theta_2^2+....+\theta_n^2}$.

If we define $\boldsymbol{\Theta}$ as $\boldsymbol{\theta}$ with the zeroth component missing, so $\boldsymbol{\Theta}$ is then an n-dimensional vector, we can say that our plane is given by the equation

$$\boldsymbol{\theta}\bullet\mathbf{x}=0 \quad\Rightarrow\quad \theta_0+\boldsymbol{\Theta}\bullet\mathbf{x}=0 \quad\Rightarrow\quad -\boldsymbol{\Theta}\bullet\mathbf{x}=\theta_o$$

The vector $-\boldsymbol{\Theta}$ is then our official normal vector for this plane, and $d=|\theta_0|\,/\,|\boldsymbol{\Theta}|$ is the distance of closest approach of this plane to the origin.

Why is this "plane" just discussed the "boundary" between Ng's famous X's and O's? We start with the plane as just described, $\boldsymbol{\theta}\bullet\mathbf{x}=0$ where $\boldsymbol{\theta}$ is the weight vector which minimizes the cost function $J(\boldsymbol{\theta})$. On one side of this plane we will have $\boldsymbol{\theta}\bullet\mathbf{x}>0$, and on the other side $\boldsymbol{\theta}\bullet\mathbf{x}<0$. We can then regard this plane as dividing our R^n space into two half spaces separated by this plane.

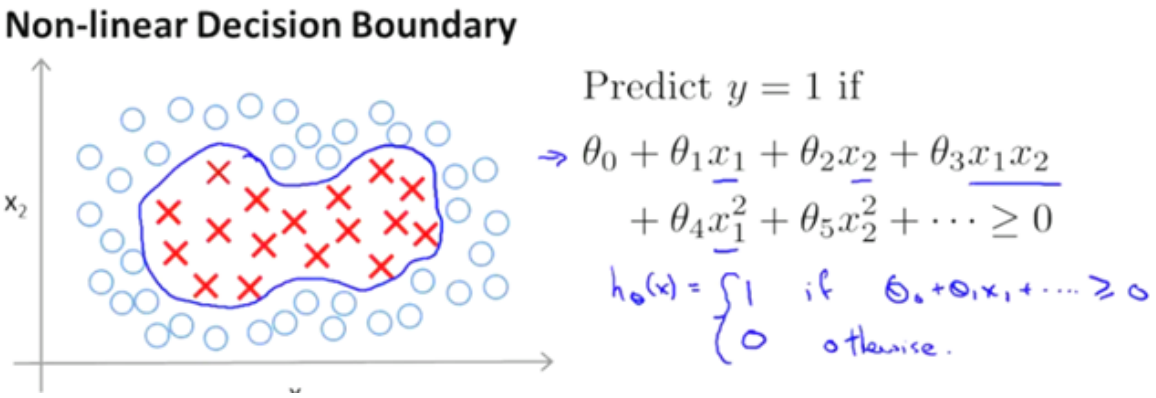
In the half space in which $\boldsymbol{\theta}\bullet\mathbf{x}>0$, we know that $g(\boldsymbol{\theta}\bullet\mathbf{x})>1/2$, so new test points $\mathbf{x}$ in this half space have $g>1/2$ and we round that up to conclude that we think $y=1$ for such a sample. This is the half space that has the trial X's (maybe not all of them, but certainly most of them).

In the half space in which $\boldsymbol{\theta}\bullet\mathbf{x}<0$, we know that $g(\boldsymbol{\theta}\bullet\mathbf{x})<1/2$, so new test points $\mathbf{x}$ in this half space have $g<1/2$ and we round that down to conclude that we think $y=0$ for such a sample. This is the half space that has the O's (maybe not all of them, but certainly most of them).

We conclude that this "boundary" arrived at by doing linear logistic regression is a "planar" boundary in $\mathcal{R}^n$. It is never going to be a curved boundary.

One could imagine, however, doing some kind of preprocessing of the features x_i into new features f_i and then using $h_\theta(\mathbf{x}) = \boldsymbol{\theta} \bullet \mathbf{f}$ in place of $\boldsymbol{\theta} \bullet \mathbf{x}$. The resulting "planar boundary" that emerges from logistic regression is still planar in f-space, but when this boundary is plotted in x-space, it might be highly curved. This is exactly what is done in the SVM algorithm where new features $f_i = f_i(\mathbf{x})$ are highly non-linear functions of $\mathbf{x}$. For example, perhaps $f_i(\mathbf{x}) = \exp[-(\mathbf{x} - \boldsymbol{\ell}^{(i)})^2/2\sigma^2]$ where vectors $\boldsymbol{\ell}^{(i)}$ are so-called landmarks. This particular feature preprocessing function is a Gaussian or normal distribution and σ is the standard deviation or "width" of this function. In the SVM algorithm, this function is referred to as a "kernel". It should be noted that in SVM, the "metric" used is a sort of "low budget" version of the metric d(s,y) described above. This simpler metric allows for fast computation.

So in general, one must use *non*-linear features like x_1^2 or $x_2x_3^3$ (non-linear logistic regression) in order to get curved separation boundaries, or one can use linear logistic regression on the abovementioned f_i which are then non-linear functions of the x_i input features. In the first case, one might end up with $h_\theta(x) = -A^2 + (x_1\text{-}a)^2 + (x_2\text{-}b)^2$ where the constants are functions of the θ_i, θ_{ij} etc type weights used, and then $h_\theta(x) = 0$ becomes $(x_1\text{-}a)^2 + (x_2\text{-}b)^2 = A^2$ and the boundary is then a circle of radius A centered at the location (a,b). The space is then partitioned by this boundary circle into two parts, and one will have $g(h_\theta(x)) > 1/2$ and will this contain the X's, and the other part will have the O's. By adding enough non-linear terms one could in theory obtain the kind of boundary that Prof. Ng likes to draw,



9. 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

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

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 logistic regression the regularization is done similarly,

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

and the derivatives then become

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

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

The purpose of doing regularization is to remove unwanted high frequency information from the fitting function so one avoids "overfitting". Prof. 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 the θ_i get "large", adding a bias therefore toward smooth functions.

Later Prof. Ng refers to overfitting as a "high-variance" situation, and underfitting as "high-bias".

10. 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 s^{th} component of the above is this:

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

The squared metric above can then be written

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

$$= (\theta \mathbf{x}^{(i)} - \mathbf{y}^{(i)})^2$$

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

$$= \Sigma_{s=1}^{K} (\Sigma_{j=0}^{n} \theta_{sj} [\mathbf{x}^{(i)}]_j - [\mathbf{y}^{(i)}]_s)^2$$

Remember that the metric $d(\mathbf{h}_\theta(\mathbf{x}), \mathbf{y})$ is a measure of the distance between the value **y** and our linear fit to the value **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),

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

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

$$= (1/2m) \Sigma_{i=1}^{m} \Sigma_{s=1}^{K} (\Sigma_{j=0}^{n} \theta_{sj} [\mathbf{x}^{(i)}]_j - [\mathbf{y}^{(i)}]_s)^2 \, .$$

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

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

This can then be compared to the scalar cost function

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

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(θ) 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 :

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

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

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

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

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

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

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

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

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

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

or

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

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

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

so that

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

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,

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

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

$$\theta^T = (X^T X)^{-1} X^T Y \qquad \text{// vector output, } \theta \text{ is a matrix}$$

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,

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

11. Modification of the Logistic 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

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

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

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

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

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

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

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

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,

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

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

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

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.

C. The Back-Propagation Algorithm for Logistic Regression

1. The Cost Function for a Neural Network

In Section B.11 the metric d and cost function J for a K-bit output classification method based on input feature vectors $\mathbf{x}^{(i)}$ were given by (the outputs are $\mathbf{y}^{(i)}$)

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

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

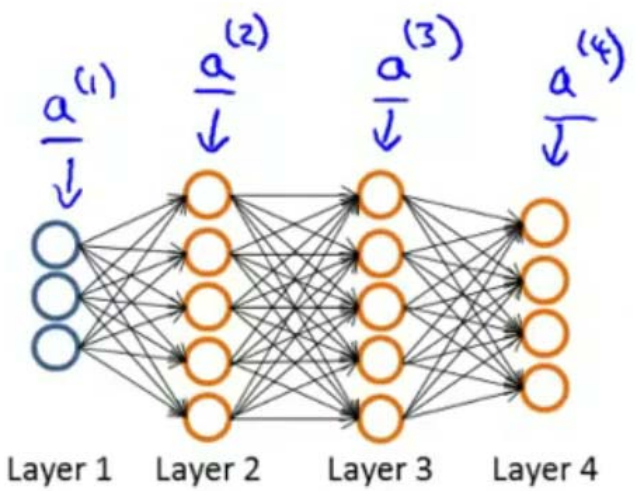
$$= -(1/m) \Sigma_{i=1}^{m} \Sigma_{k=1}^{K} [y^{(i)}_k \log([h_\theta(\mathbf{x}^{(i)})]_k) + (1-y^{(i)}_k) \log(1- [h_\theta(\mathbf{x}^{(i)})]_k)]$$

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

$\boldsymbol{\theta}^{(k)} \bullet \mathbf{x} = \Sigma_{j=0}^{n} [\boldsymbol{\theta}^{(k)}]_j x_j = \Sigma_{j=0}^{n} \theta_{kj} x_j = [\theta \mathbf{x}]_k$ // 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 Prof. 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 soft bit output from several soft bit inputs from the previous layer. Looking at the final layer, the inputs to this layer are not $\mathbf{x}$, but are $\mathbf{a}^{(3)}$. On the far left the input feature vectors $\mathbf{x}$ could be made either of hard or soft bits. On the far right, the soft output bits $a^{(4)}$ could be taken as is, or could be rounded to generate hard output bits. The inner layers of the network are sometimes called hidden layers.

Rather than refer to soft bits as soft bits all the time (ie, real numbers in the (0,1) range), we will just call them "bits" with the understanding they are normally soft bits.

The number of bits in each layer is taken to be S($\boldsymbol{\ell}$) [= 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 in Section B.11, 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

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

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

Thus we enhance the line shown above to write

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

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 (again, this is from Section B.11)

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

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

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

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 superscripts on θ and **a** match on each of the above lines, this was Prof. 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

$[\theta^{(2)}\ \mathbf{a}^{(2)}]_k = \theta^{(2)}{}_{k\alpha}\, a^{(2)}{}_\alpha$ // 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

$$J(\theta^{(L-1)},\theta^{(L-2)} \ldots\ldots \theta^{(1)}) = (1/m)\,\Sigma_{i=1}^{m}\, d(\mathbf{h_\theta}(\mathbf{a}^{(L-1,i)}),\mathbf{y}^{(i)}) \qquad \text{// general cost function}$$

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

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:

$$f(x) = a[b(c(x))] \qquad \Rightarrow \qquad f'(x) = a'[b(c(x))]\; b'(c(x))c'(x)$$

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)

$$J(\theta^{(L-1)},\theta^{(L-2)} \ldots\ldots \theta^{(1)}) = \Sigma_{i=1}^{m} d(\mathbf{h_\theta}(\mathbf{a}^{(L-1,i)}),\mathbf{y}^{(i)}) \qquad \text{// general cost function}$$

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

where

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

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 Prof. 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

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

Prof. 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".

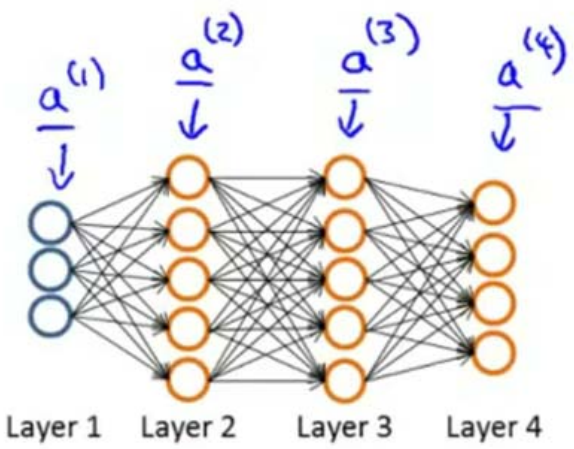
Let's go back to these results from the previous section which pertain to the Ng sample neural net studied there. The cost function is

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

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

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

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



Following Ng, we define some *new symbols* to make things more compact (each $\theta^{(\ell)}$ is a matrix)

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

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

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

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

Prof. Ng suppresses his training sample label i, so the above appears in his video 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,

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

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 $\ell = 3$:

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

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

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

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

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

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

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

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

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

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

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

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

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

Now try $\boldsymbol{\ell} = 1$ 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

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

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

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

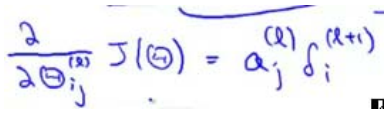
Write the above three descent derivatives in the following form,

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

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

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

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



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

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

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

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

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 $\delta^{(4)}_m$:

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

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

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

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

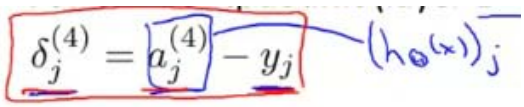
These results can be summarized as follows,

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

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

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

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



$$\delta^{(3)} = (\Theta^{(3)})^T \delta^{(4)} .* g'(z^{(3)})$$
$$\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)})$$

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 Prof. 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

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

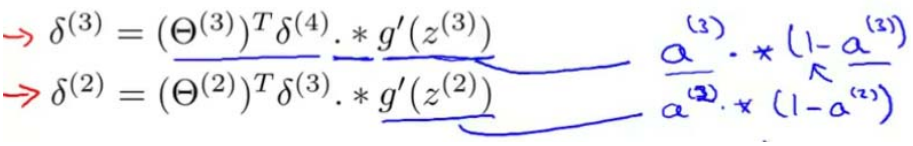
Thus for example

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

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



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

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

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

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

6. The back-propagation algorithm summarized

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 :

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

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

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

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. That step is

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

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

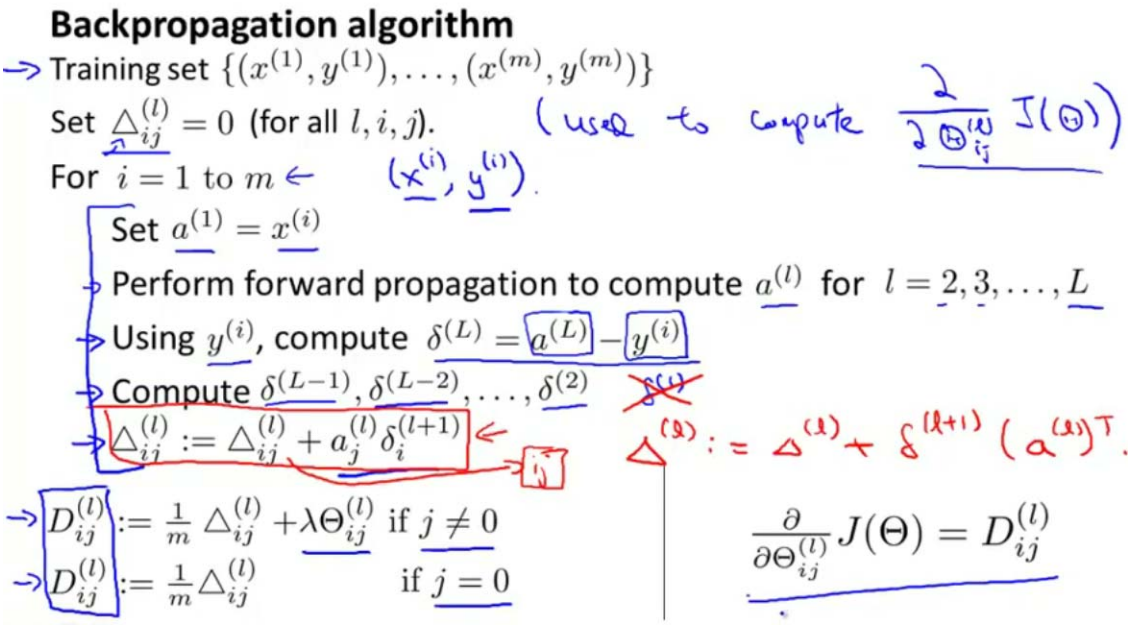
The "fifth step" is to iterate the above steps 1 through 4, watching J decrease until it stabilizes at a minimum value. Hopefully one then ends up with the "best" set of weights $\theta^{(\ell)}_{ij}$ to fit the training data.

Remember that all this is happening in the multi-output bit logistic regression framework. Once the best set of weights $\theta^{(\ell)}_{ij}$ is found from the just-described training process, one can inject a new test vector **x** into the neural net. Perhaps this is a vector of soft bit black and white pixel values of some raster image, where each pixel value lies in the range (0,1), and there are $n = N^2$ such input values for an NxN image. The neural net outputs will be the four soft bits $a^{(4)}_{k}$. If these bits mean car, person, building and tree, then one would classify input **x** according to the largest soft output bit. For example, if there are K = 4 output soft bits $a^{(4)}_{k}$ and they are { 0.1, 0.01, 0.3, .87 }, then the neural net classifier is saying it thinks the image is a tree. If the outputs are { 0.3, 0.3, 0.3, 0.3} then either the classifier has failed, or maybe none of the four prescribed objects appears in the image.

In his proposed algorithm code, Prof. 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)



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 1/x invert (for g) and this is done for each of L-1 values of k which is a small integer. It is a cheap calculation. A hardware implementation would require an exponentiater.

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.

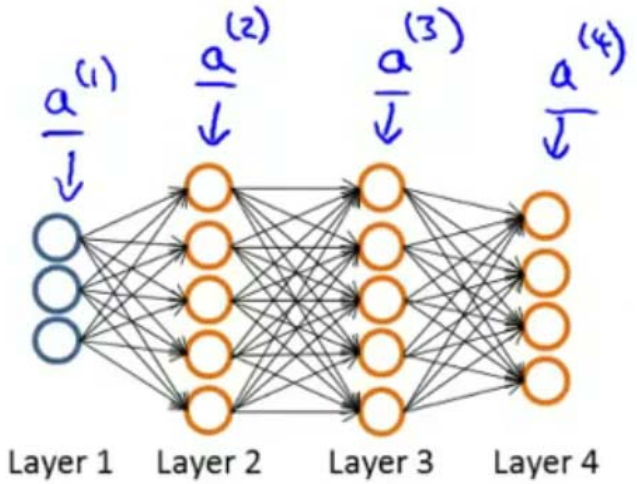
In contrast, when "gradient checking" is turned on as per video 9-5, you compute *the entire cost function* two times (+ε and -ε) for each parameter $\theta^{(\ell)}_{mn}$ and you can see that this is a long and therefore slow calculation: (notice the big sum over i, and the nesting of lots of g functions)

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

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

8. Topology comment

The network topology considered by Prof. Ng is this



I don't know if Prof. 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.

D. Notes on the Support Vector Machine Algorithm (SVM)

1. Finding a new "budget" metric for logistic regression.

As discussed in Section B.6, the particular metric used in logistic regression was this

$d(s,y) = -\log(s)$ if $y = 1$
$d(s,y) = -\log(1-s)$ if $y = 0$

which was written in the equivalent combined form

$$d(s,y) = -y \log(s) - (1-y) \log(1-s) \ .$$

This metric provides a particular measure of the distance between a hard bit y (either 0 or 1) and a soft bit s (in range 0 to 1). In the implementation of logistic regression shown earlier, we used

$$s = g(z) \qquad g(z) = 1/(1+e^{-z}) \qquad z = \boldsymbol{\theta} \bullet \mathbf{x} = \theta^T x = x^T \theta = \Sigma_{k=0}^{n} \theta_k x_k \ .$$

In terms then of variable z, we have

$$d(s,y) = -y \log(g(z)) - (1-y) \log(1- g(z))$$

$$= -y \log[1/(1+e^{-z})] - (1-y) \log[1- 1/(1+e^{-z})]$$

$$= y \{- \log[1/(1+e^{-z})]\} + (1-y) \{ -\log[e^{-z}/(1+e^{-z})]\}$$

$$\equiv \hat{d}(z,y)$$

where the hat just indicates that d and $\hat{d}$ have a different functional form on the first argument. (This notion was mentioned just before the ending summary of Section C.1 regarding h and $\hat{h}$.) So for the moment define these two functions, where we now write the log in its more standard form ln (natural log where $\ln(e^x) = x$, for example, $\ln(e^0) = \ln(1) = 0$)

$C_1(z) \equiv - \ln[1/(1+e^{-z})]$
$C_0(z) \equiv -\ln[e^{-z}/(1+e^{-z})]$

$$\hat{d}(z,y) = y\, C_1(z) + (1-y)\, C_0(z) \ .$$

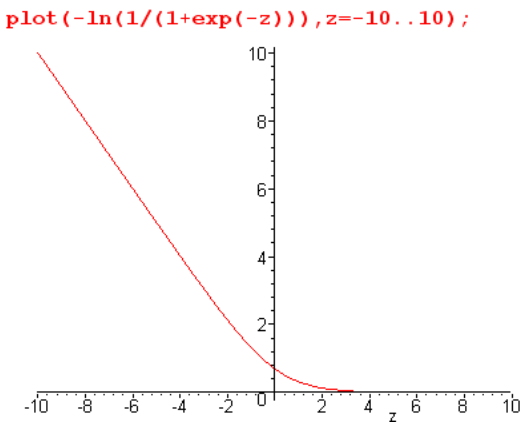
In $C_1(z)$, if z goes large negative, $e^{-z} \rightarrow \infty$ and the denominator 1 can be neglected so that

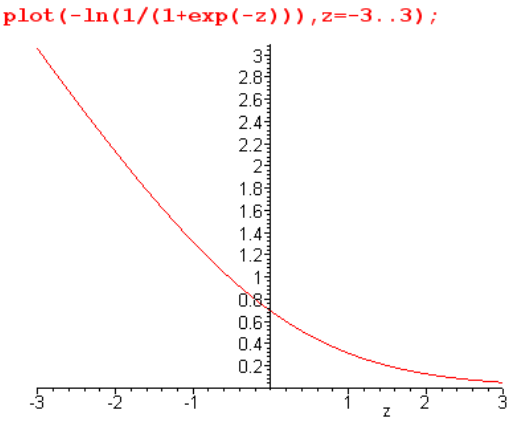
$$\lim_{z \rightarrow -\infty} C_1(z) \approx - \ln[1/e^{-z}] = -\ln(e^z) = -z \ .$$

So for large negative z, function $C_1(z)$ is asymptotic to a straight line *through the origin* of slope -1. On the other hand, for large positive z, e^{-z} is small and

$\lim_{z \to +\infty} C_1(z) \approx - \ln[1/1] = -\ln(1) = -0 = 0$

These two limits are apparent in the left Maple plot below, while the right plot shows the plot for z in the range (-3,3) which matches the plot in Prof. Ng's video 12.1



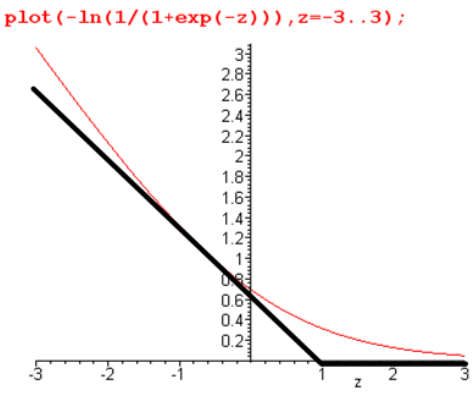


Prof. Ng next suggests trying a new metric which is similar to $\hat{d}(z,y)$ but not the same. Calling this new metric D(z,y), the idea is

$\hat{d}(z,y) = y\, C_1(z) + (1-y)\, C_0(z)$ // as above, the original metric

$D(z,y) = y\, cost_1(z) + (1-y)\, cost_0(z)$ // new metric with different functions

Here is the proposed shape of function $cost_1(z)$ (curve in heavy black)



So

$cost_1(z) = 0$ $z \geq 1$

$cost_1(z) = \alpha(z-1)$ $z \leq 1$ where α = the slope of the line shown, $\alpha < 0$

which we can combine using the Heaviside step function H(w) = 1 when w > 0 and = 0 when w < 0,

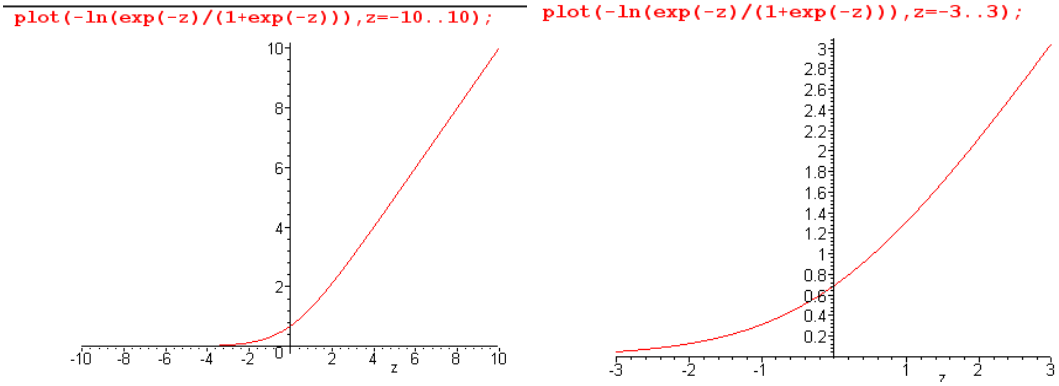
$$\text{cost}_1(z) = |\,\alpha\,|\,(1-z)\,H(1-z)$$

Since H forces 1-z > 0, $\text{cost}_1(z)$ is never negative, agreeing with the graph.

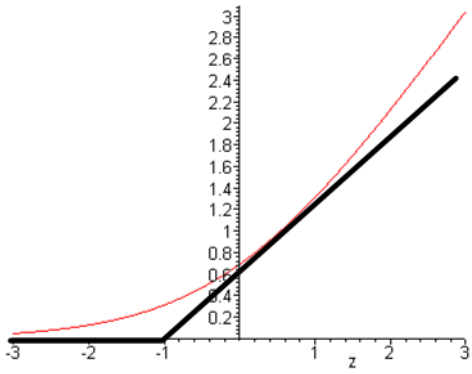
Since the asymptote of $C_1(z)$ lands at z=0 and not at the point z = 1 (see above) , this line is not going to be the asymptote of the curve $C_1(z)$, so there is some question about how to set the slope α . In the above picture where the black line is made tangent to the red $C_1(z)$ curve, it appears that α ≈ - 0.7. Probably the exact value of α is not terribly important (as we shall see below). Of course each choice gives a slightly different metric.

In similar fashion, the function $\text{cost}_0(z)$ can be defined as a budget replacement for $C_0(z)$. First here are two plots of $C_0(z)$. Similar limits to those above show that the asymptote again comes into the origin this time with slope +1, that is,

$$\lim_{z\to+\infty} C_0(z) \approx -\ln[e^{-z}/(1)] \approx -(-z) = +z$$



Then $\text{cost}_0(z)$ is the black curve shown below,



$$\text{cost}_0(z) = 0 \qquad z \le -1$$

$cost_0(z) = \beta(z+1)$ $\quad z \geq -1$ where β = the slope of the line shown, $\beta > 0$

which we can combine using the Heaviside step function H(w) = 1 when w > 0 and = 0 when w < 0,

$cost_0(z) = \beta(z+1)H(z+1)$

and again, since H forces z+1> 0, this cost is also never negative ($\beta > 0$)

Again there is a choice to make of the slope, and a rough tangent fit suggests $\beta \approx 0.6$. Notice that the two functions C_1 and C_0 are not simply horizontal reflections of each other, though the graphs make that seem to be the case. That is, $C_0(z) \approx C_1(-z)$. But the new budget functions $cost_{0,1}(z)$ can be made reflections of each other by choosing $\beta = -\alpha$. Then you would have $cost_0(z) = cost_1(-z)$.

The new metric functions $cost_{0,1}(z)$ are "budget" functions because they are much cheaper to compute than the $C_{0,1}(z)$ functions which require expo calculation(s) and then a log calculation. In the back-propagation method this was not an issue since the δ method avoided such computations, and the log and expo were only needed in the slower "error checking" diagnostic. One suspects that in the SVM method things are going to be computed directly by brute force so cheapness is good.

So here then is the new proposed "cheapo metric"

$D(z,y) = y\ cost_1(z) + (1-y)\ cost_0(z)$ $\quad z = \boldsymbol{\theta} \bullet \mathbf{x} = \theta^T x = x^T \theta$

2. The Cost Function for linear-kernel SVM (Support Vector Machine)

As Prof. Ng explains, the regularization λ is replaced by constant C which "goes the other way", the overall factor of m (trial set count) is dropped, and then here is the cost function using this new budget metric,

$J(\boldsymbol{\theta}) = C\ \Sigma_{i=1}^{m}\ D(z^{(i)},y^{(i)}) + (1/2)\ \Sigma_{j=1}^{n} \theta_j^2$

$= C\ \Sigma_{i=1}^{m}\ \{ y^{(i)}\ cost_1(z^{(i)}) + (1- y^{(i)})\ cost_0(z^{(i)})\} + (1/2)\ \Sigma_{j=1}^{n} \theta_j^2$

where now $z^{(i)} = \boldsymbol{\theta} \bullet \mathbf{x}^{(i)} = \theta^T x^{(i)}$

and here is a quote from video 12.1 (the second sum should be on j)

$$\rightarrow \min_{\theta} C \sum_{i=1}^{m} \left[y^{(i)} cost_1(\theta^T x^{(i)}) + (1 - y^{(i)}) cost_0(\theta^T x^{(i)}) \right] + \frac{1}{2} \sum_{i=1}^{n} \theta_j^2$$

The two cheap cost functions are from above (where now select $\alpha = -\beta$ where $\beta > 0$)

$cost_1(z) = \beta(1-z)\ H(1-z)$ // note that $cost_1(z) = 0$ when $z \geq 1$

$cost_0(z) = \beta(z+1)H(z+1)$. // note that $cost_0(z) = 0$ when $z \leq -1$

The cost function for the new metric is this

$$J(\boldsymbol{\theta}) = C' \sum_{i=1}^{m} \{ y^{(i)} (1- z^{(i)}) H(1- z^{(i)}) + (1- y^{(i)}) (z^{(i)}+1)H(z^{(i)}+1)\} + (1/2) \sum_{j=1}^{n} \theta_j^2$$

$$z^{(i)} = \theta^T x^{(i)} = \boldsymbol{\theta} \bullet \mathbf{x}^{(i)} \qquad C' = \beta C$$

where we just absorb slope β into the regularization factor C'. In more detail, using the dot product notation,

$$J(\boldsymbol{\theta}) = C' \sum_{i=1}^{m} \{ y^{(i)} (1- \boldsymbol{\theta} \bullet \mathbf{x}^{(i)}) H(1- \boldsymbol{\theta} \bullet \mathbf{x}^{(i)}) + (1- y^{(i)}) (\boldsymbol{\theta} \bullet \mathbf{x}^{(i)}+1)H(\boldsymbol{\theta} \bullet \mathbf{x}^{(i)}+1)\} + (1/2) \sum_{j=1}^{n} \theta_j^2 \quad .$$

Notice that the first term kicks in only when $\boldsymbol{\theta} \bullet \mathbf{x}^{(i)} < 1$ and the second only when $\boldsymbol{\theta} \bullet \mathbf{x}^{(i)} > -1$.

This is the cost function for the "linear kernel SVM method".

3. Planes of interest and an explanation of "the margin".

In regular logistic regression, the solution boundary was found to be a "plane" in $\mathcal{R}^n$ (see Section B.8) defined by the equation $\boldsymbol{\theta} \bullet \mathbf{x} = 0$ with the solution value of $\boldsymbol{\theta}$. On one side of this planar boundary, $\boldsymbol{\theta} \bullet \mathbf{x} > 0$ and $g(\boldsymbol{\theta} \bullet \mathbf{x}) > 1/2$ and this was then "the side with the X's" (training samples with $y^{(i)} = 1$). The other side had the O's. The classification of a new test sample input $\mathbf{x}$ is then determined by which half-space it lies in (in $\mathbf{x}$ "feature space") with respect to this plane $\boldsymbol{\theta} \bullet \mathbf{x} = 0$. The metric and cost function always involve the combination $\boldsymbol{\theta} \bullet \mathbf{x}^{(i)}$ where i is the training sample summation index, but the solution plane is just $\boldsymbol{\theta} \bullet \mathbf{x} = 0$.

The *reason* that the X's lie on the $\boldsymbol{\theta} \bullet \mathbf{x} > 0$ side of the planar boundary is that then their contributions to the cost function are very small: the metric d(g,y=1) is small when $g > 1/2$. Similarly, the reason the O's lie on the $\boldsymbol{\theta} \bullet \mathbf{x} < 0$ side of the boundary is exactly the same: then their contributions to the cost function are very small: the metric d(g,y=0) is small when $g < 1/2$. Of course contributions are especially small if an X or O in its correct region lies far from the separating boundary.

The logic of the previous paragraph is a bit backward, but one can state things either way. The X's and O's are of courses given up front, and the cost-minimization algorithm finds the weight vector $\boldsymbol{\theta}$ which results in a plane $\boldsymbol{\theta} \bullet \mathbf{x} = 0$ which, due to the fact that cost was minimized, has the X's on the side with $\boldsymbol{\theta} \bullet \mathbf{x} > 0$ and the O's on the side with $\boldsymbol{\theta} \bullet \mathbf{x} < 0$.

Of course this assumes that is possible to have such a plane! If the training points are intermingled so such a plane does not exist, then the regression will do the best it can and then *most* of the X's will be on one side, and *most* of the O's will be on the other side.

Now, having reviewed that situation, consider again the cost function for the linear kernel SVM,

$$J(\boldsymbol{\theta}) = C' \sum_{i=1}^{m} \{ y^{(i)} (1- \boldsymbol{\theta} \bullet \mathbf{x}^{(i)}) H(1- \boldsymbol{\theta} \bullet \mathbf{x}^{(i)}) + (1- y^{(i)}) (\boldsymbol{\theta} \bullet \mathbf{x}^{(i)}+1)H(\boldsymbol{\theta} \bullet \mathbf{x}^{(i)}+1)\} + (1/2) \sum_{j=1}^{n} \theta_j^2$$

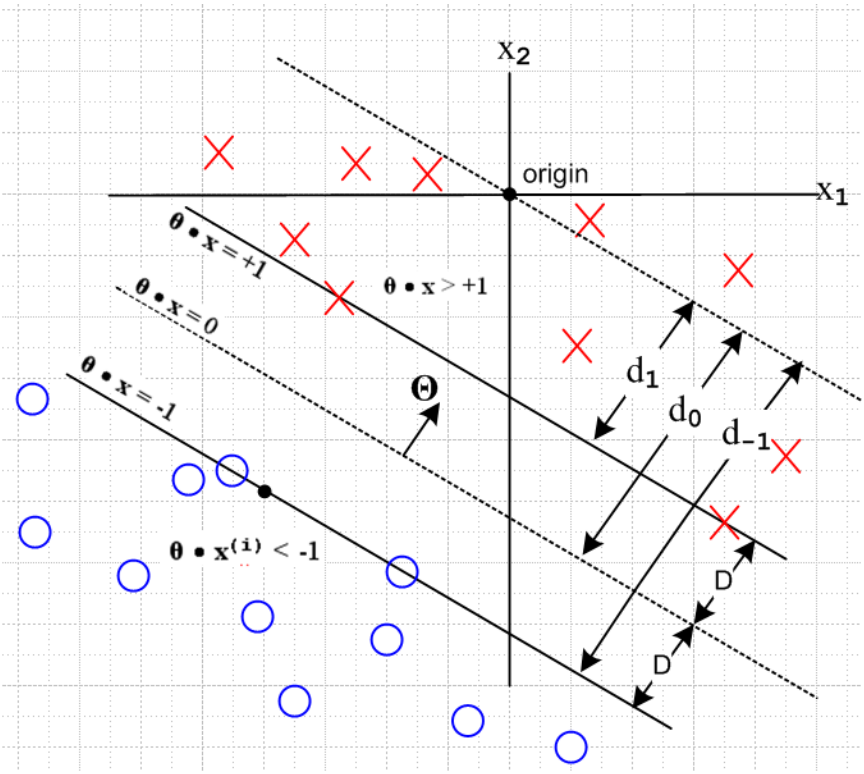
We are now going to be dealing not with one, but with two "planes", $\boldsymbol{\theta} \bullet \mathbf{x} = +1$ and $\boldsymbol{\theta} \bullet \mathbf{x} = -1$. As with simple logistic regression, it is always $\boldsymbol{\theta} \bullet \mathbf{x}^{(i)}$ which appears in the training sum, but it is $\boldsymbol{\theta} \bullet \mathbf{x}$ that appears in these plane equations (see section B.2 about equations of planes).

For $y^{(i)} = 1$ training points, it is the first term in $J(\boldsymbol{\theta})$ that is relevant (the second term vanishes). Looking only at this first term, we see that the "cost" which this term represents for a training sample is pinned to 0 by the Heaviside function when $\boldsymbol{\theta} \bullet \mathbf{x}^{(i)} > 1$ (since H(negative number) = 0). Moreover, when $\boldsymbol{\theta} \bullet \mathbf{x}^{(i)} = 1$, the cost is still 0 because H is multiplied by $(1- \boldsymbol{\theta} \bullet \mathbf{x}^{(i)})$, so it does not matter what finite value one ascribes to H(0). Therefore, there is a huge cost benefit to be had (ie, cost = 0) if the $y^{(i)} = 1$ training points lie on the $\boldsymbol{\theta} \bullet \mathbf{x}^{(i)} \geq 1$ side of the plane (which includes being on the plane). This is an especially big benefit if the regularization constant called C' above is very large (high variance, like small λ), since it magnifies the first two cost terms in $J(\boldsymbol{\theta})$.

In regular logistic regression, each training point defines its own specific plane $\boldsymbol{\theta} \bullet \mathbf{x}^{(i)} = 0$, but the aggregate process results in a plane $\boldsymbol{\theta} \bullet \mathbf{x} = 0$ which is the planar separation boundary between the X's and the O's. Just so, in the SVM method, each $y^{(i)} = 1$ training point again defines its own specific plane $\boldsymbol{\theta} \bullet \mathbf{x}^{(i)} = +1$, but in the SVM aggregate result, the boundary which the $y^{(i)} = 1$ training samples "want to be on the good side of" is going to be $\boldsymbol{\theta} \bullet \mathbf{x} = +1$ (where $\boldsymbol{\theta}$ is the solution $\boldsymbol{\theta}$). These samples want to be on the good side because that is the side which has very small cost as just noted (ie, zero cost).

Now consider the $y^{(i)} = 0$ training points. For them, it is the second term in $J(\boldsymbol{\theta})$ that counts. A training point can "achieve" a zero cost if it can find itself on the $\boldsymbol{\theta} \bullet \mathbf{x}^{(i)} \leq -1$ side of plane $\boldsymbol{\theta} \bullet \mathbf{x} = -1$ because in that case the second term's H function pins the cost to 0. As in the first case, being on the plane also has zero cost.

The result is that the solution planes $\boldsymbol{\theta} \bullet \mathbf{x} = +1$ and $\boldsymbol{\theta} \bullet \mathbf{x} = -1$ will position themselves (that is, the cost minimizing $\boldsymbol{\theta}$ will position these planes) in the following manner [as usual, we show a 2D feature space (x_1, x_2), but it should be understood that this is really an n dimensional feature space and the lines are parallel hyperplanes in that space.]



Ignoring the details for a moment, one sees that there is a "margin" or "gap" between the two solid lines in which no training points fall. This is the major characteristic of the SVM metric used above. If training points fell into the gap, such points would have a positive cost, and the SVM algorithm would increase **Θ** thus reducing D to get a lower cost (and smaller gap) and thereby remove those points from the gap. As the picture shows and the discussion notes, training points are allowed to be directly *on* the two boundaries since they still have zero cost. In the above picture one can think of such training points as "supporting" the two boundaries, and such training points, being vectors in feature space, are called "**support vectors**" , from which the algorithm "support vector machine" takes its name.

To verify the positions of these planes, we need to go back to our Section B.2 and then C.8 discussion of such planes. Again defining **Θ** as **θ** with the zeroth component missing (so **Θ** is then an n-dimensional vector) the plane **θ** • **x** = 0 may be written as [here **x** also has x_0 missing to match **Θ** but we still call it **x**]

$$\boldsymbol{\theta} \bullet \mathbf{x} = 0 \quad \Rightarrow \quad \theta_0 + \boldsymbol{\Theta} \bullet \mathbf{x} = 0 \quad \Rightarrow \quad -\boldsymbol{\Theta} \bullet \mathbf{x} = \theta_o \quad \Rightarrow \quad d_0 = |\theta_0|/|\boldsymbol{\Theta}|$$

and this means that (1) this plane has a normal vector $\mathbf{n} = \pm \boldsymbol{\Theta}$, and (2) the closest approach of this plane to the origin of feature space is $\theta_0/|\boldsymbol{\Theta}|$. For the other two planes we have

$$\boldsymbol{\theta} \bullet \mathbf{x} = +1 \quad \Rightarrow \quad \theta_0 + \boldsymbol{\Theta} \bullet \mathbf{x} = +1 \quad \Rightarrow \quad -\boldsymbol{\Theta} \bullet \mathbf{x} = \theta_o - 1 \quad \Rightarrow \quad d_1 = |\theta_0 - 1|/|\boldsymbol{\Theta}|$$

$$\boldsymbol{\theta} \bullet \mathbf{x} = -1 \quad \Rightarrow \quad \theta_0 + \boldsymbol{\Theta} \bullet \mathbf{x} = -1 \quad \Rightarrow \quad -\boldsymbol{\Theta} \bullet \mathbf{x} = \theta_o + 1 \quad \Rightarrow \quad d_{-1} = |\theta_0 + 1|/|\boldsymbol{\Theta}|$$

Since all three planes have the same normal **Θ,** they are all parallel. The closest origin approach distances are shown above. Therefore, the spacing D is given by (here we assume $|\theta_0| > 1$ to match the picture but the conclusion $D = 1/|\boldsymbol{\Theta}|$ is true for any θ_0)

$D = d_0 - d_1 = d_{-1} - d_0 = 1/|\mathbf{\Theta}|$,

which means the spacing between the two separation planes $\mathbf{\theta} \bullet \mathbf{x} = +1$ and $\mathbf{\theta} \bullet \mathbf{x} = -1$ is $2/|\mathbf{\Theta}|$.

Recall now the SVM cost function from above,

$$J(\mathbf{\theta}) = C' \sum_{i=1}^{m} \{ y^{(i)} (1 - \mathbf{\theta} \bullet \mathbf{x}^{(i)}) H(1 - \mathbf{\theta} \bullet \mathbf{x}^{(i)}) + (1 - y^{(i)}) (\mathbf{\theta} \bullet \mathbf{x}^{(i)} + 1) H(\mathbf{\theta} \bullet \mathbf{x}^{(i)} + 1) \} + (1/2) \sum_{j=1}^{n} \theta_j^2 \ .$$

The regularization term in the SVM cost function is $(1/2) \sum_{j=1}^{n} \theta_j^2 = (1/2) |\mathbf{\Theta}|^2$. This is *quadratic* in the θ_i values, whereas the first two terms are only linear in θ_i (or they are zero if the H is right). Thus, the $J(\mathbf{\theta})$ minimizer is going to favor a small value of $|\mathbf{\Theta}|$ and that in turn means a large value of the margin D shown above, since $D = 1/|\mathbf{\Theta}|$. In fact, the algorithm will push both lines outward until they both touch some support vectors, since such pushing out maintains zero cost in the first two terms of J and at the same time reduces $|\mathbf{\Theta}|$, thus reducing the regularization cost. For this reason, SVM is also called the "maximum margin separator" algorithm.

It is possible that even when D = 0, one *cannot find* a separator plane because the training points are too intermingled. In this case, presumably SVM would produce some finite width gap and some training points would be inside the gap and perhaps some even on the wrong side of the gap.

Comment: One may recall Prof. Ng saying that the vector $\mathbf{\theta}$ is normal to the SVM dividing plane, and one might have wondered why that was the case. This fact falls out naturally from the above discussion, but the normal vector is $\mathbf{\Theta}$ which is $\mathbf{\theta}$ without the θ_0 component. See the picture.

4. Arm-waving argument for why the cost function is "convex" so no local minima.

The cost function shown above is this

$$J(\mathbf{\theta}) = C' \sum_{i=1}^{m} \{ y^{(i)} (1 - \mathbf{\theta} \bullet \mathbf{x}^{(i)}) H(1 - \mathbf{\theta} \bullet \mathbf{x}^{(i)}) + (1 - y^{(i)}) (\mathbf{\theta} \bullet \mathbf{x}^{(i)} + 1) H(\mathbf{\theta} \bullet \mathbf{x}^{(i)} + 1) \} + (1/2) \sum_{j=1}^{n} \theta_j^2$$

The step function H(w) has a derivative $\delta(w)$ where δ is the famous Dirac delta function which vanishes everywhere away from w = 0. The chain rule says that $\partial/\partial w \, [f(w)H(w)] = f'(w)H(w) + f(w)\delta(w)$. If we agree to "stay away from" the region where w = 0, we can ignore this second term.

So, consider the derivative $\partial J(\mathbf{\theta})/\partial \theta_j$ of the above expression for the cost function. If we ignore those δ functions (by having trial points stay away from the two parallel hyperplanes), then the resulting derivative (ignoring the regularization term) is a constant independent of $\mathbf{\theta}$. This is so because the first line above is linear in θ_j . If we then compute the second derivative $\partial J(\mathbf{\theta})/\partial \theta^2_j$ we get 0 from the first line, and we get a positive constant (1/2) from the regularization term. This says that the "curvature" of the function $J(\theta_j)$ plotted against θ_j is positive and is independent of θ_j and this is true for all components θ_j. That means $J(\theta_j)$ is "cupping up" everywhere. A function which cups up everywhere cannot have inflection points, and so cannot have local minima.

Hopefully there is a grain of truth in this admittedly wobbly argument.

5. What does the SVM cost function look like in θ-space ? (origami)

Consider the contribution to the cost function from a $y^{(i)} = 1$ training point (ignoring constants)

$$J(\boldsymbol{\theta}) = (1 - \boldsymbol{\theta} \bullet \mathbf{x}^{(i)})\, H(1 - \boldsymbol{\theta} \bullet \mathbf{x}^{(i)}) \ .$$

Ignoring the H, this has the form (assume a 2D feature space)

$$J(\theta_1, \theta_2) = 1 - \theta_0 - \theta_1 x^{(i)}{}_1 - \theta_2 x^{(i)}{}_2$$

Write this as

$$J = 1 - \theta_0 - a\theta_1 - b\theta_2 \qquad a = x^{(i)}{}_1 \qquad b = x^{(i)}{}_2$$

which can be rewritten as

$$a\theta_1 + b\theta_2 + J = (1 - \theta_0) \ .$$

In the 3D space whose coordinates are (θ_1, θ_2 , J) this equation describes a tilted plane (see Section B.2). The normal to this plane is given by $\mathbf{n}$ = (a,b,1) and this plane's closest approach to the origin of the 3D space is $d = |1-\theta_0|/n$ where $n = \sqrt{a^2+b^2+1}$.

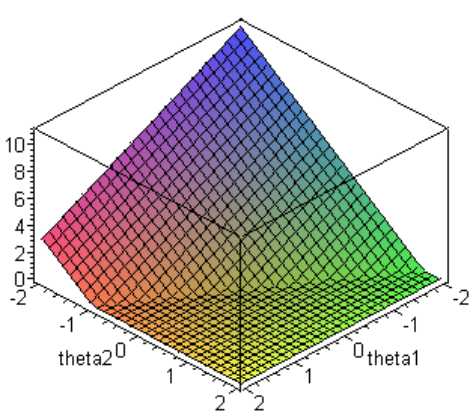
It seems clear, by the way, that if you add two such planes, the sum is a new plane which has a normal obtained by adding the normals of the two planes:

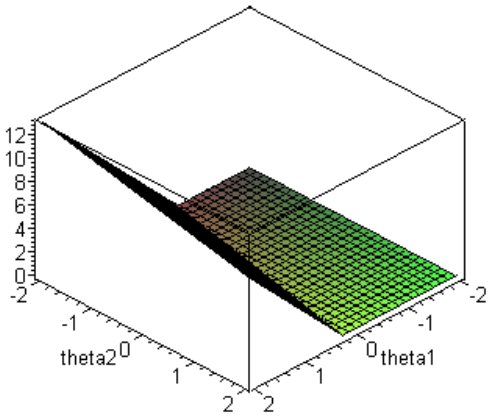
$J(\theta_1, \theta_2) = (1-\theta_0) - \theta_1 a - \theta_2 b$	normal = (a,b,1)
$K(\theta_1, \theta_2) = (1-\theta_0) - \theta_1 c - \theta_2 d$	normal = (c,d,1)
$L(\theta_1, \theta_2) = J+K = 2(1-\theta_0)_0 - \theta_1(a+c) - \theta_2(b+d)$	normal = (a+c, b+d,1)

Such an addition of planes would happen if we considered two terms in the J(θ) sum, perhaps the first two training values with i = 1 and i = 2.

However, with regard to J($\boldsymbol{\theta}$) above, the H factor causes this plane to be "bent" so that part of the plane lies at J = 0. For example, here might be the planes corresponding to two trial points for $\mathbf{x}^{(1)}$ = (2,3) and $\mathbf{x}^{(2)}$ = (-5,1), where we set $\theta_0 = 0$:

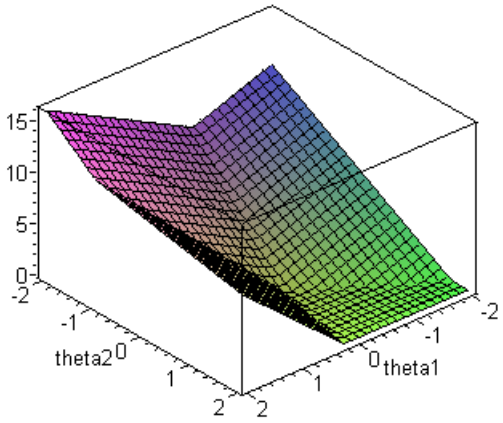
```
f1 := (1 - 2*theta1 - 3*theta2)*Heaviside(1 - 2*theta1 - 3*theta2):
f2 := (1 + 5*theta1 - theta2)*Heaviside(1 + 5*theta1 - theta2):
plot3d(f1, theta1 = -2..2, theta2=-2..2, grid = [30,30], axes = boxed):
plot3d(f2, theta1 = -2..2, theta2 = -2..2, grid = [30,30], axes = boxed):
```

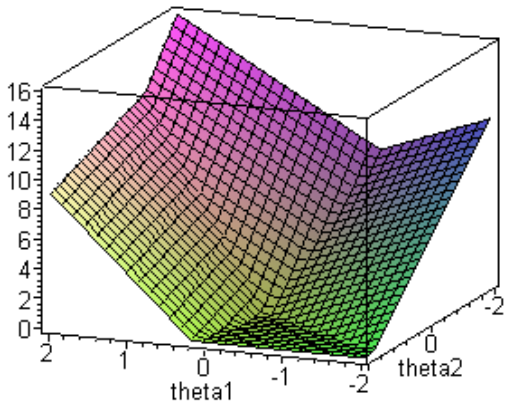




When these two bent planes are added in the trial sum over i, the result is a sort of origami figure which has some portion flattened on the bottom and has three distinct planar areas, the new one in the region where the two planes are additive. Here are two views of the two added bent planes:

```
plot3d(f1+f2, theta1 = -2..2, theta2=-2..2, grid = [30,30], axes = boxed);
```





The mostly purple plane is the new one where there is addition of the two planes (neither being 0).

So this is an example of the origami surface obtained by adding just two trial terms in the sum which have $y^{(i)} = 1$. One can imagine that trial terms with $y^{(i)} = 0$, being represented by the second term in our sum, will result in similar origami surfaces. Suppose there are 500 trial points. One can then imagine that the sum of all these planar surfaces is an extremely complicated surface having many flat faces which meet at many linear "seams".

Despite all these seams (where the gradient and normal are really undefined), it still seems *possible* that one could do "gradient descent". For example, if a descent step crosses a seam, one might end up zigzagging down the seam, but at least one is on the move. But if the step size is large compared to the size of the little flat patches, there will likely be trouble.

6. Doing the SVM minimization

In section 4 above, it was noted that computing $\partial J/\partial \theta_j$ derivatives is going to result in nasty delta functions associated with all the seams of the complicated origami-shaped J cost function. Any one who

knows about delta functions would say that such computations are probably a bad path to follow. They are an indicator of potential numerical trouble lying in the road ahead.

Once again, here is the cost function :

$$J(\boldsymbol{\theta}) = C' \sum_{i=1}^{m} \{ y^{(i)} (1 - \boldsymbol{\theta} \bullet \mathbf{x}^{(i)}) H(1 - \boldsymbol{\theta} \bullet \mathbf{x}^{(i)}) + (1 - y^{(i)}) (\boldsymbol{\theta} \bullet \mathbf{x}^{(i)} + 1) H(\boldsymbol{\theta} \bullet \mathbf{x}^{(i)} + 1) \} + (1/2) \sum_{j=1}^{n} \theta_j^2$$

This thing is pretty "cheap" in computation terms since it is just a sum of a bunch of dot products added or subtracted from 1.

$\boldsymbol{\theta} \bullet \mathbf{x}^{(i)} = \theta^T x^{(i)}$ = a fast "vectorized" calculation using a vector dot product library routine

The H functions turn this minimization problem in to a sort of "linear programming" problem due to all the seams and many flat surfaces on which J is constrained. A brute force algorithm would just "vary" the θ_i components in some intelligent heuristic way and for each candidate $\boldsymbol{\theta}$ would compute $J(\boldsymbol{\theta})$ as shown above and sees if it is more or less than the lowest J obtained from previous variations. Eventually it has explored all of the relevant parts of $\boldsymbol{\theta}$ space and gets no more reduction in J and then halts and declares victory. The lack of local minima noted in section 4 above rules out a false victory.

In practice, the SVM minimizing code minimizes $|\boldsymbol{\Theta}|$ (thus maximizing the gap between the lines, $2/|\boldsymbol{\Theta}|$ as noted above) subject to inequality conditions that all training points have zero cost, which means all training points have non-positive arguments for the relevant H functions shown above. That means

$y^{(i)} = 1$	$H(1 - \boldsymbol{\theta} \bullet \mathbf{x}^{(i)})$	$1 - \boldsymbol{\theta} \bullet \mathbf{x}^{(i)} \le 0$	$\boldsymbol{\theta} \bullet \mathbf{x}^{(i)} \ge 1$	$\boldsymbol{\Theta} \bullet \mathbf{x}^{(i)} + \theta_0 \ge 1$
$y^{(i)} = 1$	$H(\boldsymbol{\theta} \bullet \mathbf{x}^{(i)} + 1)$	$1 + \boldsymbol{\theta} \bullet \mathbf{x}^{(i)} \le 0$	$\boldsymbol{\theta} \bullet \mathbf{x}^{(i)} \le -1$	$\boldsymbol{\Theta} \bullet \mathbf{x}^{(i)} + \theta_0 \le -1$.

This simply means that no training points are allowed to fall inside the gap between the two lines, which is the way our picture above is drawn. One can change the y symbol to be

$$Y = 2y - 1$$

which means when y = 1, Y = 1 as before, but when y = 0, Y = -1, so then we have Y = ± 1 as the two classifications. Then both inequalities above can be trivially written as a single inequality,

$Y^{(i)} [\boldsymbol{\Theta} \bullet \mathbf{x}^{(i)} + \theta_0] \ge 1$ $\quad$ i = 1,2,3....m $\quad$ m = # training samples

The minimization problem is then this:

minimize $|\boldsymbol{\Theta}|$ $\quad$ subject to $\quad$ $Y^{(i)} [\boldsymbol{\Theta} \bullet \mathbf{x}^{(i)} + \theta_0] \ge 1$ $\quad$ for $\quad$ i = 1,2,3....m .

If we now do this change of symbols,

$Y^{(i)} \rightarrow y_i$ $\quad$ $\mathbf{x}^{(i)} \rightarrow \mathbf{x}_i$ $\quad$ $\boldsymbol{\Theta} \rightarrow \mathbf{w}$ $\quad$ $\theta_0 \rightarrow -b$, $\quad$ $m \rightarrow n$

the minimization problem can be written as

minimize $|\mathbf{w}|$ subject to $y_i[\ \mathbf{w} \bullet \mathbf{x}_i - b\ ] \geq 1$ for $i = 1,2,3....n$

and this is the way the problem is stated in wiki

http://en.wikipedia.org/wiki/Support_vector_machine .

As wiki mentions, the solution of this kind of generic minimization problem can be efficiently implemented by minimizing $|\mathbf{w}|^2/2$ (to avoid square roots) and by using the traditional method of Lagrange multipliers (a set of constants α_i) in a quadratic programming context. In this approach, the weight vector $\mathbf{w}$ is expanded as a linear combination of the products of training values y_i and $\mathbf{x}_i$ which lie on (or very close to) the separating planes (ie, these are the support vectors)

$\mathbf{w} = \Sigma_i\ \alpha_i y_i\ \mathbf{x}_i$.

Since most training values don't lie on the planes, there are a small number of terms in this sum and thus only a small number of α_i to figure out.

Prof. Ng properly suggested that students accept the SVM algorithm as a black box technique and not get side-tracked into all the technology involved in its implementation.

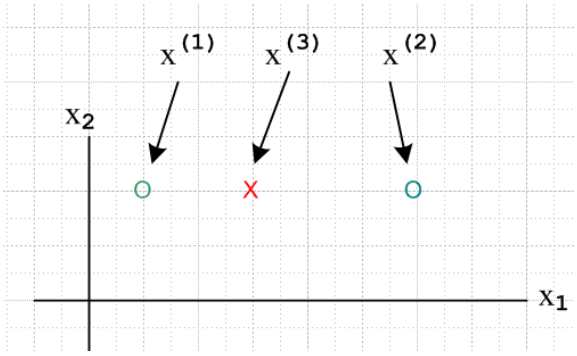
7. The SVM kernels

As described above, the no-kernel SVM method can only find a *planar* separation boundary strip. If one is willing to preprocess the raw training features x_i into some "smarter" features f_i using non-linear functions, then the resulting separation planes in the new f_i feature space, when mapped back into x_i feature space, appear as "parallel" curved separation surfaces. These non-linear preprocessing functions are called kernels and this idea was described at the end of Section B.8 above with respect to regular logistic regression.

Here is the basic question. Assume there are a bunch of X's and O's in x-feature-space, such as Ng's pattern

Are there a set of non-linear functions $f_i(\mathbf{x})$ for i = 1,2...J (J as yet unknown) for which a separating *plane* in f-space will map into a successful separating *curve* in x-space (like the blue curve above)? I think some theorem says that the answer is yes, and in particular there are successful function sets $f_i(\mathbf{x})$ with J = m, the number of training samples.

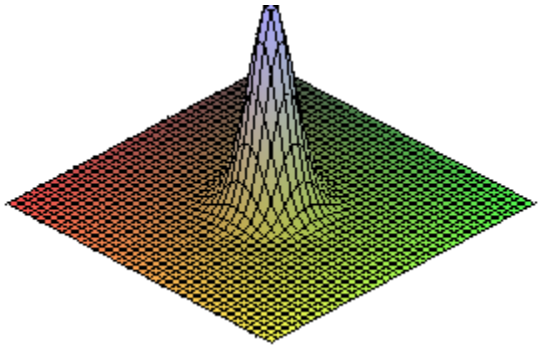
Here is a specific example which hopefully demonstrates the reasonableness of this theorem. Suppose there are only m = 3 training samples and only n = 2 features so $x = (x_1, x_2)$ for each training sample, and suppose the three $x^{(i)}$ values look like this in x-space,



There is no linear separating plane in x-space for this example because the three points are collinear with the different value in the middle. Now suppose we try J = m = 3 with these three Gaussian functions,

$f_1(\mathbf{x}) = \exp(- | \mathbf{x} - \mathbf{x}^{(1)}|^2/2\sigma^2)$
$f_2(\mathbf{x}) = \exp(- | \mathbf{x} - \mathbf{x}^{(2)}|^2/2\sigma^2)$
$f_3(\mathbf{x}) = \exp(- | \mathbf{x} - \mathbf{x}^{(3)}|^2/2\sigma^2)$

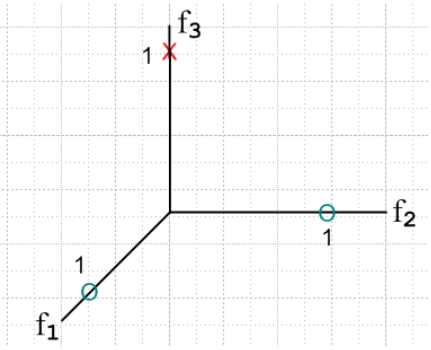
and suppose we assume that the standard deviation σ is small so these functions are highly peaked around $\mathbf{x} \approx \mathbf{x}^{(i)}$. Here is a plot of $f_3(\mathbf{x})$ centered at $\mathbf{x}^{(3)}$ for some small σ,



In passing, notice that the equation $f_3(x_1,x_2) = 0.2$ would describe a perfect circle in the x_1,x_2 plane centered at $\mathbf{x}^{(3)}$ since it would imply that $| \mathbf{x} - \mathbf{x}^{(3)}|$ = constant in the exponent, and this is the distance between $\mathbf{x}$ and $\mathbf{x}^{(3)}$. Here then are some simple facts:

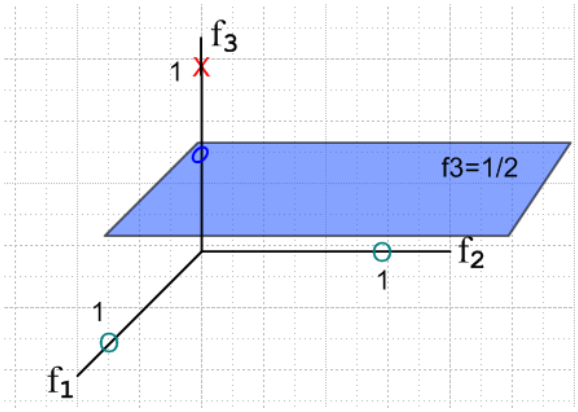
The point $\mathbf{x} = \mathbf{x}^{(1)}$ in x-space maps into a point f = (1,ε,ε') in f-space, where the ε's are close to 0.
The point $\mathbf{x} = \mathbf{x}^{(2)}$ in x-space maps into a point f = (ε,1,ε') in f-space, where the ε's are close to 0.
The point $\mathbf{x} = \mathbf{x}^{(3)}$ in x-space maps into a point f = (ε,ε',1)) in f-space, where the ε's are close to 0.

So roughly speaking, here are the locations of the X's and O's in f-space,



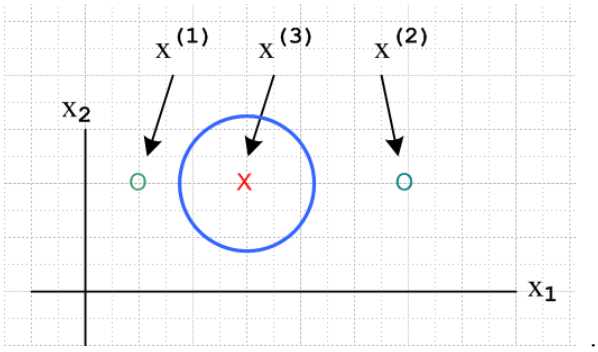
Each point is close to the tip of a unit vector in f-space. Notice the "f-space" is really a cube of edge 1 because each f lies in the range (0,1) for the Gaussian functions shown above.

If we ran SVM or logistic regression in f-space as described above, we would have no trouble coming up with a nice separating plane in f-space. The optimal plane would probably be one which is normal to the triangle connecting the three points shown, but let's instead assume a simpler dividing plane which is $f_3 = 1/2$:



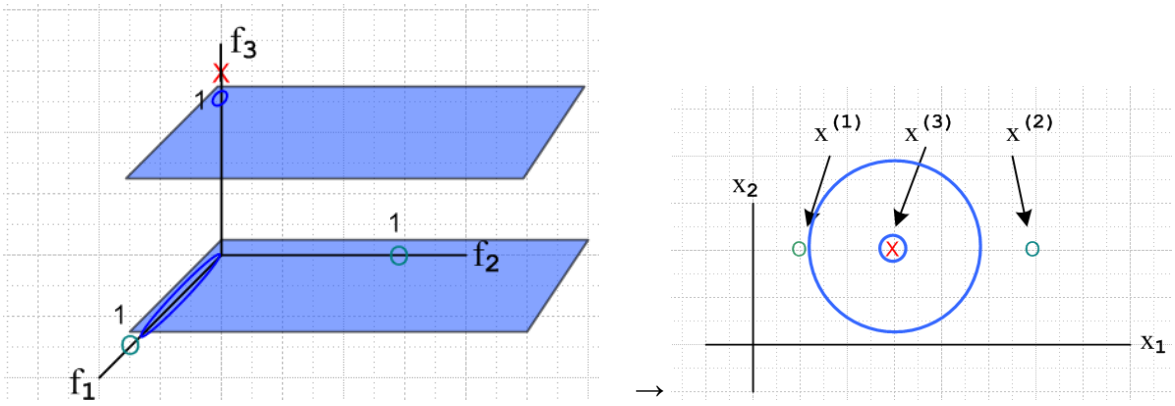
The red X lies above this plane, the two green O's lie below it. This plane is $\boldsymbol{\theta} \bullet \mathbf{f} = 0$ with a somewhat suboptimal solution weight vector $\boldsymbol{\theta}$. Since $f_0 = 1$, this says $\theta_0 + \boldsymbol{\Theta} \bullet \mathbf{f} = 0$ in earlier notation, which says $\boldsymbol{\Theta} \bullet \mathbf{f} = -\theta_0$. Thus, based on Section B.2, we have $\theta_0 = -1/2$ (distance from origin is 1/2) and $\boldsymbol{\Theta}$ is the normal vector which then points up or down.

Back in x-space, this dividing plane maps into $1/2 = \exp(- | \mathbf{x} - \mathbf{x}^{(3)}|^2/2\sigma^2)$ which we already noted is a circle in x-space. So here is what the (now successful) dividing surface looks like in x-space,



To be more precise, the blue circle in x-space maps into a curve that lies on the $f_3 = 1/2$ plane in f-space. The mapping $f_i(\mathbf{x}) = \exp(- | \mathbf{x} - \mathbf{x}^{(i)}|^2/2\sigma^2)$ maps each point in x-space to a point in f-space, so any 2D curve in x-space must map into a curve in f-space (not a surface in f space), and circles centered on the red X map into curves in f space which lie on some f_3 = constant plane. If the blue circle is far from the green O's, then the back-mapped curve in f space lies near location $f_1=f_2= 0$ on an $f_3=$ constant plane.

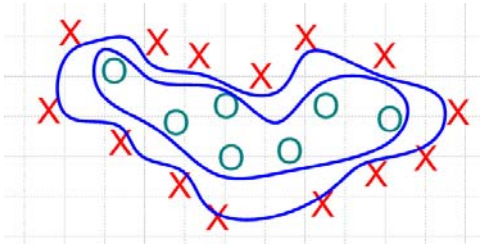
The SVM in fact returns two dividing planes with maximum gap, and we can pretend for our simplified example that these two planes are at $f_3 = .01$ and $f_3 = .99$ where the gap is about 2D = 1. Then



Each of these planes in f-space corresponds to a circle in x-space in the sense described in the previous paragraph. The two curves in x-space are "parallel" in the sense that there is an empty gap between them and they are similar in nature (here, concentric circles). Since all points on the larger blue circle are far from the point $\mathbf{x}^{(2)}$, in f-space all points on the back-mapped curve will be near $f_2 = 0$. Some points on this circle are close to $\mathbf{x}^{(1)}$ and for those points $f_1 \approx 1$, but most points on the circle are far from $\mathbf{x}^{(1)}$ and for these points $f_1 \approx 0$. For the small blue circle, all points are far from both $\mathbf{x}^{(1)}$ and $\mathbf{x}^{(2)}$ so in f-space this small circle maps into a very small closed curve near $f_1= 0$ and $f_2= 0$ on the $f_3 = .99$ plane. We have made a crude attempt to draw both the back-mapped closed curves on their respective planes in f space. The lower curve in f-space is far from the green O on the f_2 axis, and correspondingly the blue circle in x-space stays far from $\mathbf{x}^{(2)}$. Our suboptimal f-space dividing planes produce suboptimal dividing curves in x-space for our example.

This example can be extended to some arbitrary number m of training samples. In this case, the m training points in x-space will map near the tips of the m unit vectors in m-dimensional f space. Based on which

points are X's and which are O's, the SVM will find a maximal-gap pair of parallel separating planes in this m dimensional f space, and that will map into a pair of separating curves in x space between which no training samples will fall:



E. Notes on Principle Component Analysis (PCA)

This subject involves several different mathematical ingredients which will be described below and then mixed together at the end to derive the PCA method.

1. Continuous Probability Density Functions and Moments

Imagine an experiment involving some element of randomness that is performed a trillion times and reports out a value of some continuous variable x which lies in range $(-\infty,\infty)$. These trillion values of x are placed in bins of width Δx. Then $N_{\mathbf{x}}\,\Delta x$/ trillion = the probability that x is in the bin $(x,x+\Delta x)$, where $N_{\mathbf{x}}$ is the count in this bin. As the number of experiments $\rightarrow \infty$, and the bin width $\rightarrow$ dx, one finds that the probability that the reported-out x will lie in the bin (x,x+dx) is given by some p(x)dx where p(x) is some non-negative continuous function called the probability density function or pdf for short. Since the total probability of reporting out any value must be 1, we have

$$\int_{-\infty}^{\infty} dx\ p(x) = 1\ .$$

It is traditional to refer to such a random variable as capital X, as if it were some different entity (a "random variable") which takes values x according to p(x). So X is the random variable and x are the values which that random variable takes.

A joint probability function p(x,y)dxdy is the probability that X has a value in (x,x+dx) *and* that Y has a value in the range (y,y+dy). The normalization is given in this case by

$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} dxdy\ p(x,y) = 1$$

Integrating p(x,y) over just y gives the probability that x lies in (x,x+dx),

$$p(x)dx = \int_{-\infty}^{\infty} p(x,y)\,dxdy = dx * \int_{-\infty}^{\infty} dy\ p(x,y)$$

$$\Rightarrow \qquad p(x) = \int_{-\infty}^{\infty} dy\ p(x,y)\ .$$

Example: In quantum mechanics, $p(\mathbf{x}) = |\psi(\mathbf{x})|^2$ describes the probability that a particle described by the "wavefunction" $\psi(x)$ lies in a volume $p(\mathbf{x})\ d^3\mathbf{x}$.

The **mean value** of a random variable X associated with p(x) is

$$\mu_{\mathbf{x}} = \int_{-\infty}^{\infty} x\,p(x)\,dx = E(X),$$

where the notation E(X) means that μ_x is the "Expected value" of X, the value X is most likely to have. One can compute other "moments" of the distribution p(x) such as "the n^{th} moment"

$$E(X^n) = \int_{-\infty}^{\infty} x^n \, p(x) \, dx$$

or even some complicated joint moment

$$E(X^n Y^m) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} dx \, dy \, x^n \, y^m \, p(x,y) \, .$$

All these moments change as one varies the location of the origin, something not always desirable. That leads one to consider instead the so-called "central moments" like

$$\int_{-\infty}^{\infty} (x-\mu_x)^n \, p(x) \, dx$$

and these don't change as the origin changes. For example, suppose we shift the origin to the left by distance a, so that $p(x) \rightarrow p(x-a)$ and $\mu_x \rightarrow \mu_x + a$. Perhaps p(x) was peaked at x = 0, it is now peaked at x = a. Perhaps the mean was 0, and it is now a. Then

$$\int_{-\infty}^{\infty} (x-\mu_x)^n \, p(x) \, dx \;\rightarrow\; \int_{-\infty}^{\infty} (x-[\mu_x+a])^n \, p(x-a) \, dx$$

Let x' = x-a and dx = dx' so

$$\int_{-\infty}^{\infty} (x-\mu_x-a)^n \, p(x-a) \, dx = \int_{-\infty}^{\infty} (x'-\mu_x)^n \, p(x') \, dx' \;=$$ same as before shifting the origin.

2. Variance and Standard Deviation

The second central moment has a special name, it is called "the variance of random variable X" :

$$\text{var}(X) = \int_{-\infty}^{\infty} (x-\mu_x)^2 \, p(x) \, dx. \qquad \text{where} \quad \mu_x = \int_{-\infty}^{\infty} x \, p(x) \, dx \, .$$

Notice that the variance is always a positive number.

Example: The normalized "Gaussian" pdf is given by (normalized meaning $\int_{-\infty}^{\infty} p(x) \, dx = 1$)

$$p(x) = (1/\sqrt{2\pi}\,\sigma)\exp[-(x-\mu)^2/2\sigma^2]$$

and if this is inserted into the integrals above one finds that

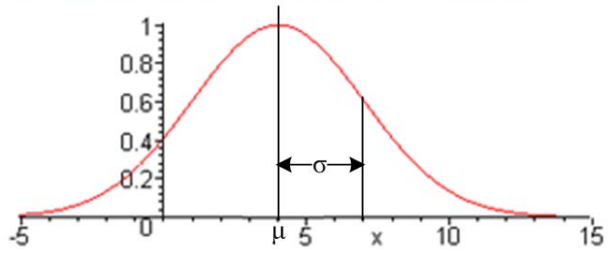
$\mu_x = \mu$ // mean

$\text{var}(X) = \sigma^2$ // variance, σ = standard deviation

The variance is a measure of how spread out ("dispersed") p(x) is . For large σ, p(x) is a wide bell curve, while for small σ p(x) is highly peaked around its mean μ.

For any distribution p(x), the square root of the variance is called "the standard deviation", so for the Gaussian with parameter σ as shown, σ is the standard deviation. So σ is a rough measure of the half width of a distribution. Here is a (non-normalized) Gaussian with $\mu = 4$ and $\sigma = 3$,

```
plot(exp(-(x-4)^2/(2*3^2)), x=-5..15);
```



A frequently used fact is the following,

$\sigma_x^2 \equiv \text{var}_x = E(X^2) - \mu_x^2$.

Proof: $\text{var}_x = E([X-\mu_x]^2) = E(X)^2 - 2\mu_x E(X) + \mu_x^2 E(1) = E(X^2) - 2\mu_x^2 + \mu_x^2 = E(X^2) - \mu_x^2$.

3. Covariance and Correlation

The covariance between two continuous random variables is the natural generalization of the variance for one variable : (we now dispense with integration endpoints)

$$\text{cov}(X,Y) = \int\int (x-\mu_X)\,(y-\mu_Y)\; p(x,y)\, dx\, dy$$

$$\mu_X = \int\int x\, p(x,y)\, dx\, dy$$

$$\mu_Y = \int\int y\, p(x,y)\, dx\, dy \quad .$$

If X and Y are the same variable, then p(x,y) = 0 when $x \neq y$ and in fact $p(x,y) = p(x)\delta(x-y)$ and one gets

$$\text{cov}(X,X) = \int\int (x-\mu_X)\,(y-\mu_X)\, p(x)\delta(x-y)\, dy = \int (x-\mu_X)^2\, p(x)\, dx = \text{var}(X)$$

so the covariance of a variable with itself is just the variance, and so cov(X,X) is a positive number. Recall that the delta function has this famous "sifting property" which is used in the above line,

$$\int dy\ f(y)\ \delta(x-y) = f(x).$$

The interpretation of cov(X,Y) is this: if the quantities $(x-\mu_X)$ and $(y-\mu_Y)$ tend to have the same sign as one views the space over which p(x,y) is defined , then the covariance is a positive number, and this indicates that the two variables are somehow "correlated". If Y tends to be negative when X is positive, then cov(X,Y) will be a negative number and things are still "correlated". If cov(X,Y) ≈ 0, then the two random variables have little or no correlation. If the pdf p(x,y) = r(x)s(y), the variables are independent, and one expects that cov(X,Y) = 0,

$$\text{cov}(X,Y) = \int\int (x-\mu_X)\,(y-\mu_Y)\,r(x)s(y)\,dx\,dy = \int dx\,(x-\mu_X)\,r(x) \ * \int dy\ (y-\mu_Y)s(y)\,dy$$

$$= (\mu_X - \mu_x) * (\mu_Y - \mu_Y) = 0 * 0 = 0\ .$$

The covariance is the "central moment version" of E(XY)

$$\text{cov}(X,Y) = \int\int (x-\mu_X)\,(y-\mu_Y)\ p(x,y)\,dx\,dy$$

$$E(XY) = \int\int x\,y\ p(x,y)\,dx\,dy\ .$$

A related quantity is the correlation of X and Y which is the covariance scaled down by the product of the two standard deviations

$$\text{corr}(X,Y) \equiv \text{cov}(X,Y)/(\sigma_x\sigma_y) \qquad \sigma_x^2 = \text{var}_x = \int_{-\infty}^{\infty} (x-\mu_x)^2\ p_x(x)\,dx$$

$$\sigma_y^2 = \text{var}_y = \int_{-\infty}^{\infty} (y-\mu_y)^2\ p_y(y)\,dy$$

4. Mean, Variance and Covariance for Discrete Distributions

One can imagine a continuous probability density function p(x,y,z) being 0 everywhere except at m tiny clots near certain points $\mathbf{x}^{(i)}$ in the x,y,z space. As these clots are made very small, we can think of them as delta functions. To get into the Ng notation, let's use x_1,x_2,x_3 instead of x,y,z. Consider then,

$$p(x_1,x_2,x_3) = \Sigma_{i=1}^{m} P(\mathbf{x}^{(i)})\ \delta(x_1 - x^{(i)}_1)\ \delta(x_2 - x^{(i)}_2)\ \delta(x_3 - x^{(i)}_3) = \Sigma_{i=1}^{m} P(\mathbf{x}^{(i)})\delta^{(3)}(\mathbf{x} - \mathbf{x}^{(i)}).$$

This is how one can write the continuous pdf when the probability of our variables being in the clot located at $\mathbf{x}^{(i)}$ is $P(\mathbf{x}^{(i)})$. If one were to integrate the above over all three variables x_1, x_2, x_3 the result has to be 1, and one would conclude from doing so the reasonable fact that $\Sigma_{i=1}^{m} P(\mathbf{x}^{(i)}) = 1$.

In the Ng videos, where one sees a set of X's located in some x_1, x_2 plane, one implication is that all those X's are regarded as equally likely. Each training sample counts as much as any other. So from now on we can just set $P(\mathbf{x}^{(i)}) = 1/m$ and then $\Sigma_{i=1}^{m} P(\mathbf{x}^{(i)}) = \Sigma_{i=1}^{m} (1/m) = (1/m)\ m = 1$, as required. So the continuous pdf in the Ng discrete world is given by

$$p(x_1,x_2,x_3) = (1/m)\,\Sigma_{i=1}^{m}\,\delta(x_1-x^{(i)}_1)\,\delta(x_2-x^{(i)}_2)\,\delta(x_3-x^{(i)}_3) \;=\; (1/m)\,\Sigma_{i=1}^{m}\,\delta^{(3)}(\mathbf{x}-\mathbf{x}^{(i)}),$$

and we are just summing over all the probability clots located at the various training sample points. The corresponding expressions for $p(x_1)$ and $p(x_1,x_2)$ are exactly what one would think based on $p(x_1,x_2,x_3)$ above,

$$p(x_1,x_2) = (1/m)\,\Sigma_{i=1}^{m}\,\delta(x_1-x^{(i)}_1)\,\delta(x_2-x^{(i)}_2) \;=\; (1/m)\,\Sigma_{i=1}^{m}\,\delta^{(2)}(\mathbf{x}-\mathbf{x}^{(i)})$$
$$p(x_1) \;\;= (1/m)\,\Sigma_{i=1}^{m}\,\delta(x_1-x^{(i)}_1)$$

So we can now insert the above into the various continuous pdf results above to see what happens.
For the mean, one gets

$$\mu_{x1} = \int_{-\infty}^{\infty} x_1\,p(x_1)\,dx_1 \;=\; \int_{-\infty}^{\infty} x_1\,(1/m)\,\Sigma_{i=1}^{m}\,\delta(x_1-x^{(i)}_1) \;= (1/m)\,\Sigma_{i=1}^{m}\,x^{(i)}_1$$

which just says the mean is what everyone knows the mean should be. Similarly for μ_{x2} with $x^{(i)}_2$.

For the variance we have

$$\mathrm{var}(X_1) = \int_{-\infty}^{\infty} (x_1-\mu_{x1})^2\,p(x_1)\,dx_1 = \int_{-\infty}^{\infty} (x_1-\mu_{x1})^2\,(1/m)\,\Sigma_{i=1}^{m}\,\delta(x_1-x^{(i)}_1)$$

$$= (1/m)\,\Sigma_{i=1}^{m}\,(x_1^{(i)} - \mu_{x1})^2$$

and this certainly is what one would expect as the variance of a discrete set of equally likely samples. It is a measure of how much they are spread out in the x_1 direction. There is some different variance in the X_2 direction

$$\mathrm{var}(X_2) = (1/m)\,\Sigma_{i=1}^{m}\,(x^{(i)}_2 - \mu_{x2})^2$$

One might be interested in the "total variance" summed over all dimensions of feature space,

$$\mathrm{var}\,(\mathbf{X}) = \Sigma_{s=1}^{n}\,\{(1/m)\,\Sigma_{i=1}^{m}\,(x^{(i)}_s - \mu_{Xs})^2\} \;= (1/m)\,\Sigma_{i=1}^{m}\,\Sigma_{s=1}^{n}(x^{(i)}_s - \mu_{Xs})^2$$

$$= \;(1/m)\,\Sigma_{i=1}^{m}\,(\mathbf{x}^{(i)} - \boldsymbol{\mu}_X)\bullet(\mathbf{x}^{(i)} - \boldsymbol{\mu}_X) \;= (1/m)\,\Sigma_{i=1}^{m}\,\|\,\mathbf{x}^{(i)} - \boldsymbol{\mu}_X\,\|^2\,.$$

Finally, the covariance for a discrete distribution is given by

$$\mathrm{cov}(X_1,X_2) = \int\int\,(x_1-\mu_{Xa})\,(x_2-\mu_{x2})\;\;p(x_1,x_2)\,dx_1\,dx_2$$

$$= \int\int\,(x_1-\mu_{Xa})\,(x_2-\mu_{x2})\;\{\,(1/m)\,\Sigma_{i=1}^{m}\,\delta(x_1-x^{(i)}_1)\,\delta(x_2-x^{(i)}_2)\}\;\;dx_1\,dx_2$$

$$= (1/m)\,\Sigma_{i=1}^{m} \int\int (x_1-\mu_{x_a})\,(x_2-\mu_{x_2})\,\{\,\delta(x_1-x^{(i)}{}_1)\,\delta(x_2-x^{(i)}{}_2)\}\;\; dx_1\,dx_2$$

$$= (1/m)\,\Sigma_{i=1}^{m}\; [\int (x_1-\mu_{x_a})\,\{\,\delta(x_1-x^{(i)}{}_1)\,dx_1\,][\int (x_2-\mu_{x_2})\,\delta(x_2-x^{(i)}{}_2)\}\;\; dx_2]$$

$$= (1/m)\,\Sigma_{i=1}^{m}\; [(x^{(i)}{}_1-\mu_{x_1})\,][\,(x^{(i)}{}_2-\mu_{x_2})]\;\; .$$

If we have lots of random variables X_1, X_2.....X_n (as we do in feature space), we could examine the covariance between any pair of them (say X_a and X_b),

$$\mathrm{cov}(X_a,X_b) = (1/m)\,\Sigma_{i=1}^{m}\; (x^{(i)}{}_a - \mu_{x_a})\,(x^{(i)}{}_b - \mu_{x_b})$$

One can regard this as a matrix with two indices a and b, so write

$$\Sigma_{ab} \equiv \mathrm{cov}(X_a,X_b) \;\; = \;\; (1/m)\,\Sigma_{i=1}^{m}\; (x^{(i)}{}_a - \mu_{x_a})\,(x^{(i)}{}_b - \mu_{x_b})\;.$$

The diagonal element Σ_{aa} is just the regular variance for coordinate x_a ,

$$\Sigma_{aa} \equiv \mathrm{cov}(X_a,X_a) \;\; = \;\; (1/m)\,\Sigma_{i=1}^{m}\; (x^{(i)}{}_a - \mu_{x_a})^2 = \mathrm{var}(X_a)\;.$$

Notice that $(x^{(i)}{}_a - \mu_{x_a})$ is the a^{th} component of the vector $(\mathbf{x}^{(i)}-\boldsymbol{\mu}_x)$ where $\mathbf{x}^{(i)}$ is the training point and where $\boldsymbol{\mu}_x$ is the vector of means over features x_1, x_2....x_n . Then the above matrix can be written in full matrix notation as

$$\Sigma = \;(1/m)\,\Sigma_{i=1}^{m}\; (\mathbf{x}^{(i)}-\boldsymbol{\mu}_x)\,(\mathbf{x}^{(i)}-\boldsymbol{\mu}_x)^T$$

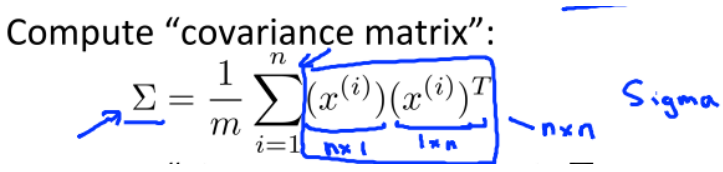
If this seems strange, consider this matrix made of two vectors **a** and **b** (for n=2 dimensions)

$$M = \mathbf{a}\,\mathbf{b}^T \;=\; \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} (b_1\;b_2\;) \;=\; \begin{pmatrix} a_1b_1 & a_1b_2 \\ a_2b_1 & a_2b_2 \end{pmatrix}.$$

Prof. Ng suggested that the feature data should be both pre-scaled and pre-adjusted so that all the means are zero, which is to say, such that $\boldsymbol{\mu}_x = 0$. *Then* the above covariance matrix becomes

$$\Sigma = \;(1/m)\,\Sigma_{i=1}^{m}\, \mathbf{x}^{(i)}\;\mathbf{x}^{(i)T}$$

which is the way it appears in the videos and slides, namely (upper endpoint on sum should be m),



As is clear from its definition (using zero-mean adjusted data),

$\Sigma_{ab} \equiv (1/m)\, \Sigma_{i=1}^{m}\, x^{(i)}{}_a\, x^{(i)}{}_b$,

the matrix Σ is real and symmetric, $\Sigma_{ab} = \Sigma_{ba}$. Matrices which are real and symmetric have certain mystical properties which we need to now review.

5. Real symmetric matrices: the eigenvalue problem and diagonalization of M into D

So let M be some nxn real symmetric matrix. A major theorem of linear algebra says that any such matrix can be "brought to diagonal form" by transformation with some real orthogonal matrix R in this manner:

$D = R^{-1}MR \qquad D_{ij} = \delta_{i,j} D_{ii} \qquad D = \mathrm{diag}(D_{11}, D_{22}, \ldots D_{nn})$.

A real orthogonal matrix is one for which $R^T = R^{-1}$ (or $RR^T = R^T R = 1$ = the unit matrix). Since $RR^T = 1$, one knows that

$1 = \det(1) = \det(R)\det(R^T) = \det(R)\,\det(R) = [\,\det(R)]^2 \qquad \Rightarrow \qquad \det(R) = \pm 1.$

Real orthogonal matrices with det(R) = 1 are simply rotation matrices (albeit in n dimensions). Those with det(R) = -1 are products of rotation matrices times matrices which reflect axes. We don't care about these right now but an example will appear below.

Eigenvalues and eigenvectors. Any symmetric matrix like M or D has eigenvalues and eigenvectors (another theorem of linear algebra),

$M\mathbf{u}^{(i)} = \lambda_i \mathbf{u}^{(i)}$ $\quad \lambda_i$ = eigenvalues $\quad \mathbf{u}^{(i)}$ = eigenvectors
$D\mathbf{e}^{(i)} = \lambda'_i \mathbf{e}^{(i)}$. $\quad \lambda'_i$ = eigenvalues $\quad \mathbf{e}^{(i)}$ = eigenvectors .

Assuming for the moment that this claim is true, one sees that the eigenvectors are undetermined by an arbitrary scale factor. If $\mathbf{u}^{(i)}$ is an eigenvector, then so is 5 $\mathbf{u}^{(i)}$. To remove this scale ambiguity, we normalize the eigenvectors and use unit vector eigenvectors $\hat{\mathbf{u}}^{(i)} \equiv \mathbf{u}^{(i)} / \| \mathbf{u}^{(i)} \|$, so then

$M\hat{\mathbf{u}}^{(i)} = \lambda_i \hat{\mathbf{u}}^{(i)}$ $\quad \lambda_i$ = eigenvalues $\quad \hat{\mathbf{u}}^{(i)}$ = normalized eigenvectors
$D\hat{\mathbf{e}}^{(i)} = \lambda'_i \hat{\mathbf{e}}^{(i)}$ $\quad \lambda'_i$ = eigenvalues $\quad \hat{\mathbf{e}}^{(i)}$ = normalized eigenvectors

In the second case, it is easy to write down the eigenvalues and eigenvectors, and one finds,

$\lambda'_i = D_{ii}$ and $\hat{\mathbf{e}}^{(i)}{}_j = \delta_{i,j}$.

For example, $\hat{\mathbf{e}}^{(1)} = (1,0,0...)$, $\hat{\mathbf{e}}^{(2)} = (0,1,0..0)$ and so on. To prove this claim, consider

$D\,\hat{\mathbf{e}}^{(i)} = \lambda'_i\, \hat{\mathbf{e}}^{(i)}$ // statement of EV equation

$\Sigma_b D_{ab}\, \hat{e}^{(i)}{}_b = \lambda'_i\, \hat{e}^{(i))}{}_a$ // component "a" of the above equation

$\Sigma_b [D_{aa}\delta_{a,b}] [\delta_{i,b}] = \lambda'_i[\delta_{i,a}]$

$D_{aa} [\delta_{i,a}] = \lambda'_i[\delta_{i,a}] = \lambda'_a[\delta_{i,a}]$ => $D_{aa} = \lambda'_a$.

Now consider this set of operations:

$D\,\hat{\mathbf{e}}^{(i)} = \lambda'_i\,\hat{\mathbf{e}}^{(i)}$

$(R^{-1}MR)\,\hat{\mathbf{e}}^{(i)} = \lambda'_i\,\hat{\mathbf{e}}^{(i)}$ // now apply R from the left on both sides

$(MR)\,\hat{\mathbf{e}}^{(i)} = \lambda'_i R\,\hat{\mathbf{e}}^{(i)}$

$M(R\hat{\mathbf{e}}^{(i)}) = \lambda'_i(R\,\hat{\mathbf{e}}^{(i)})$

But comparison of this last equation with the eigenvalue equation for M,

$M\hat{\mathbf{u}}^{(i)} = \lambda_i\hat{\mathbf{u}}^{(i)}$,

tells us two facts:

(1) $\lambda'_i = \lambda_i$ // M and D have the same eigenvalues

(2) $\hat{\mathbf{u}}^{(i)} = R\hat{\mathbf{e}}^{(i)}$ // how eigenvectors of M are related to those of D.

The second line tells us an interesting fact,

$\hat{u}^{(i)}{}_a = \Sigma_b R_{ab}\,\hat{\mathbf{e}}^{(i)}{}_b = \Sigma_b R_{ab}\,\delta_{i,b} = R_{ai}$.

Since the second index on R_{ai} labels the columns of R_{ai}, this last line says that the vectors $\hat{\mathbf{u}}^{(i)}$ are the columns of matrix R

$R = [\,\hat{\mathbf{u}}^{(1)}, \hat{\mathbf{u}}^{(2)}, \ldots.\, \hat{\mathbf{u}}^{(n)}\,]$.

Principal Axes. The transformation $D = R^{-1}MR$ which brings real symmetric matrix M into diagonal form D is sometimes called the **principal axis transformation**, and the $\hat{\mathbf{u}}^{(i)}$ are the **principal axes**. We will be carrying out this operation with the covariance tensor, but there are many other examples where this subject arises. One is the inertia tensor of an extended rigid object, and another is the stress tensor which describes forces inside and on the surface of a continuous flexible material like jello or steel or water.

Comment. The principal axis subject is often treated not in terms of $D = R^{-1}MR$ but in terms of M and D sandwiched between vectors $\mathbf{x}$, so that one then has $m(\mathbf{x}) = \mathbf{x}^T M\mathbf{x}$ and $d(\mathbf{x}) = \mathbf{x}^T D\mathbf{x}$ where m and d are

then polynomials of degree 2 in the x_i known as **quadratic forms**. Since D is diagonal, d(**x**) has no cross terms like x_2x_3 and only terms like $\lambda_2 x_2^2$. The equation m(x) = constant describes some weirdly rotated ellipsoid in $\mathcal{R}^n$ and the equation d(x) = constant describes this same ellipsoid viewed from a reference frame in which the axes (our $\hat{\mathbf{u}}^{(i)}$) align with the obvious "principal axes" of the ellipsoid.

$d(x) = \Sigma_i \lambda_i x_i^2 = 1$ => $1/\sqrt{\lambda_i}$ are the semi-major ellipsoidal axes

Finding the eigenvalues. Given real symmetric matrix M, of course D and R are not yet known. The method of finding the λ_i and $\mathbf{u}^{(i)}$ for M is as follows. First, write $M\mathbf{u}^{(i)} = \lambda_i \mathbf{u}^{(i)}$ in this form, where now I is the unit matrix,

$(M-\lambda I)\,\mathbf{u}^{(i)} = \mathbf{0}$. λ stands for one of the eigenvalues

If the matrix (M-λI) had an inverse $(M-\lambda I)^{-1}$, then $\mathbf{u}^{(i)} = (M-\lambda I)^{-1}\mathbf{0} = \mathbf{0}$ which says there are no eigenvectors except **0**. Since we know that is not the case, it must be that (M-λI) has no inverse, and since $A^{-1} = \text{cof}(A^T)/\det(A)$ for any square matrix A, it must be that det(M-λI) = 0. This baby is called the secular or characteristic equation and it is a polynomial of degree n in the variable λ. If this polynomial is factored, then det(M-λI) = 0 says $(\lambda-\lambda_1)(\lambda-\lambda_2)\ldots(\lambda-\lambda_n) = 0$ where the λ_i are functions of the M_{ij}. This is how you find the eigenvalues λ_i of the matrix M. You write out this secular equation, factor it, and there you are. Or you have Maple or some other program just tell you the answer.

Finding the eigenvectors. Having the eigenvalues, the next step is to find the eigenvector for each one. For each eigenvalue λ_i, the equation $M\mathbf{u}^{(i)} = \lambda_i \mathbf{u}^{(i)}$ in components is

$\Sigma_{b=1}^{n} M_{ab} u^{(i)}{}_b = \lambda_i u^{(i)}{}_a$ a = 1,2...n

This is n linear equations in the n unknowns $u^{(i)}{}_a$ which we know can be solved using something like "Cramer's Rule" from high school algebra. From $\mathbf{u}^{(i)}$ one then gets the $\hat{\mathbf{u}}^{(i)} = \mathbf{u}^{(i)} / |\mathbf{u}^{(i)}|$.

So that is how, given M, one can find the λ_i and the $\hat{\mathbf{u}}^{(i)}$. Once one has the $\hat{\mathbf{u}}^{(i)}$, one then knows the matrix R which brings M to diagonal form D: $R = [\hat{\mathbf{u}}^{(1)}, \hat{\mathbf{u}}^{(2)}, \ldots \hat{\mathbf{u}}^{(n)}]$. And of course one knows that the diagonal elements of D are $D_{ii} = \lambda_i$.

Orthonormal basis. The eigenvectors $\hat{\mathbf{u}}^{(i)}$ of a real symmetric matrix M have another important property as yet not mentioned: they are orthonormal, meaning $\hat{\mathbf{u}}^{(i)} \bullet \hat{\mathbf{u}}^{(j)} = \delta_{i,j}$, which we will show in a moment. Our simple proof given below requires that the eigenvalues all be different from each other, but if some of the eigenvalues are the same, one can still find $\hat{\mathbf{u}}^{(i)}$ which are orthogonal by a procedure known as Gram-Schmidt Orthogonalization (GSO) which is one of the painful little aspects of linear algebra which we shall ignore, though it is really quite simple in concept.

The conclusion is that the eigenvectors $\hat{\mathbf{u}}^{(i)}$ form an orthogonal basis for space $\mathcal{R}^n$. One can think of the set $\{\hat{\mathbf{u}}^{(i)}\}$ as a set of rotated orthogonal coordinate axes sitting in $\mathcal{R}^n$.

Here then is the proof of orthogonality under the assumption stated above: (here $i \neq j$)

$M\,\hat{u}^{(i)} = \lambda_i\,\hat{u}^{(i)}$ // now apply $\hat{u}^{(j)T}$ on the left

$M\,\hat{u}^{(j)} = \lambda_j\,\hat{u}^{(j)}$ // now apply $\hat{u}^{(i)T}$ on the left

$\hat{u}^{(j)T} M\,\hat{u}^{(i)} = \lambda_i \hat{u}^{(j)T}\,\hat{u}^{(i)} = \lambda_i \hat{u}^{(j)} \bullet \hat{u}^{(i)}$

$\hat{u}^{(i)T} M\,\hat{u}^{(j)} = \lambda_j \hat{u}^{(i)T}\,\hat{u}^{(j)} = \lambda_j \hat{u}^{(j)} \bullet \hat{u}^{(i)}$ // $a \bullet b = a^T b = b^T a$

$\hat{u}^{(j)T} M\,\hat{u}^{(i)} - \hat{u}^{(i)T} M\,\hat{u}^{(j)} = (\lambda_i - \lambda_j)\,\hat{u}^{(j)} \bullet \hat{u}^{(i)}$ // subtract previous two lines.

But the left side vanishes because its two terms are the same :

$\hat{u}^{(j)T} M\,\hat{u}^{(i)} = \Sigma_{a,b}\,\hat{u}^{(j)}{}_a M_{ab}\,\hat{u}^{(i)}{}_b$

$= \Sigma_{a,b}\,\hat{u}^{(i)}{}_b M_{ab}\,\hat{u}^{(j)}{}_a$ // reordering the three factors

$= \Sigma_{b,a}\,\hat{u}^{(i)}{}_a M_{ba}\,\hat{u}^{(j)}{}_b$ // swapping the names of the summation indices $a \leftrightarrow b$

$= \Sigma_{b,a}\,\hat{u}^{(i)}{}_a M_{ab}\,\hat{u}^{(j)}{}_b$ // because M is symmetric

$= \hat{u}^{(i)T} M\,\hat{u}^{(j)}$.

Therefore we have

$0 = (\lambda_i - \lambda_j)\,\hat{u}^{(j)} \bullet \hat{u}^{(i)}$.

Since $\lambda_i \neq \lambda_j$ (by our assumption of all different), we must have $\hat{u}^{(j)} \bullet \hat{u}^{(i)} = 0$. And of course in the case that i = j we have $\hat{u}^{(i)} \bullet \hat{u}^{(i)} = \|\,\hat{u}^{(i)}\|^2 = 1$ since these are unit vectors.

Invariance of the trace. The trace of a square matrix is the sum of the diagonal elements,

$\mathrm{tr}(M) = \Sigma_i M_{ii}$.

Consider $D = R^{-1} M R$ where R is real orthogonal and maybe D is not diagonal because R is the wrong R. In any event we have (implicit sums on repeated indices)

$\mathrm{tr}(D) = D_{ii} = R^T{}_{ij} M_{jk} R_{ki} = (R_{ki} R^T{}_{ij})\, M_{jk} = \delta_{k,j}\, M_{jk} = M_{jj} = \mathrm{tr}(M)$.

In the case that D is diagonal with eigenvalues λ_i, the trace is $\mathrm{tr}(D) = \Sigma_{i=1}^{n} \lambda_i$. Once upon a time, the trace was called the spur.

Example: If M is a covariance matrix Σ, and D is a diagonalized covariance matrix Σ', then the sum of the "variances" along the diagonal of Σ (namely $\Sigma_{i=1}^{n} \Sigma_{ii}$) equals the same sum for Σ' (namely $\Sigma_{i=1}^{n} \lambda_i$). Thus under a rotation the total variance summed on all axes does not change. This seems reasonable in

that the total variance of some 3D splatter of dots should be the same for observers rotated with respect to each other, just as it is for observers translated with respect to each other.

Getting diagonal elements into decreasing order. Imagine that we have diagonalized symmetric matrix M into $D = R^{-1}MR$, where D has diagonal elements λ_i . If the elements λ_i are not in decreasing order along the diagonal, one can adjust R to make this be the case. Let the adjusted R be $R' = RR_1$. Then

$$D' = R'^{-1}MR' = (RR_1)^{-1}M\,(RR_1) = R_1^{-1}(R^{-1}MR)R_1 = R_1^{-1}DR_1$$

so the only problem is to find some real orthogonal R_1 which gets the diagonal elements of D into descending order in D'. Since R and R_1 are real orthogonal, R' will be as well:

$$R'^{-1}R' = (RR_1)^{-1}\,(RR_1) = R_1^{-1}(R^{-1}R)R_1 = R_1^{-1}R_1 = 1\ .$$

Here is a simple Maple example:

```
D := matrix(3,3,[a,0,0,0,b,0,0,0,c]);
```

$$D := \begin{bmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & c \end{bmatrix}$$ suppose $b \geq a \geq c$

```
R1 :=matrix (3,3,[0,1,0,1,0,0,0,0,1]);
```

$$R1 := \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$ shuffling matrix R_1

```
evalm(inverse(R1) &* R1);
```

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$ $R_1^{-1}R_1 = 1$ so R_1 is real orthogonal => (RR_1) also real orthogonal

```
Dp := evalm(inverse(R1) &* D &* R1);
```

$D' = R_1^{-1}DR_1$

$$Dp := \begin{bmatrix} b & 0 & 0 \\ 0 & a & 0 \\ 0 & 0 & c \end{bmatrix}$$ now diagonal elements are in the right order

The diagonal elements of M_{ij} are all $\leq \lambda_1$ of D. Here we assume that in D the eigenvalues are in descending order down the diagonal so λ_1 is the largest eigenvalue

$$\lambda_1 \geq \lambda_2 \geq \lambda_3 \ldots\ldots \geq \lambda_n\ .$$

Then from the relation $D = R^{-1}MR$ we have $RDR^{-1} = M$ so that (implied sums on j and k but not on i !),

$$M_{ii} = (RDR^{-1})_{ii} = (RDR^{T})_{ii} = R_{ij}D_{jk}R^{T}_{ki} = R_{ij}\delta_{k,j}\lambda_j R^{T}_{ki} = R_{ij}\lambda_j R^{T}_{ji}$$

$= \lambda_j (R_{ij})^2 = \Sigma_{j=1}^{n} \lambda_j (\hat{\mathbf{u}}^{(j)}{}_i)^2$

where in the last step we used a result from section 5 above. Since $(\hat{\mathbf{u}}^{(j)}{}_i)^2$ is the squared i^{th} component of the j^{th} unit vector, it must be some number less than 1, call it f_j. Then we have

$M_{ii} = \Sigma_{j=1}^{n} \lambda_j f_j = \lambda_1 f_1 + \lambda_2 f_2 + \lambda_3 f_3 .... \lambda_n f_n$

$\leq \lambda_1 f_1 + \lambda_2 f_2 + \lambda_2 f_3 .... \lambda_2 f_n = \lambda_1 f_1 + \lambda_2 [f_2 + f_3 .... f_n] = \lambda_1 f_1 + \lambda_2 [1-f_1]$

$= \lambda_1 + \lambda_1 (f_1 - 1) + \lambda_2 [1-f_1] = \lambda_1 - (1-f_1)(\lambda_1 - \lambda_2) \leq \lambda_1$

where in the last step we note that since the λ_i are sorted, $(\lambda_1 - \lambda_2) > 0$ and of course $(1-f_1) > 0$ as well. Therefore

$M_{ii} \leq \lambda_1 \quad i = 1,2...n$ // no sum on i

This result is true whether or not M is positive definite (see section 7 below), it just has to be symmetric.

Example: If M is the covariance matrix Σ, then when it is diagonalized into Σ' with descending diagonal elements λ_i, one will have $\Sigma_{ii} \leq \lambda_1$. This says none of the variances of Σ (its diagonal elements) can exceed the maximum variance shown in Σ' (λ_1).

6. A little about tensors and frames of reference: an interpretation of a diagonalized matrix

This section is done in a certain standard notation which will then get slightly modified in section 8.

One thinks of two spaces we will call x-space and x'-space, and, in our case of interest, a point in x'-space is related to a point in x-space in this simple manner,

$x'_a = R_{aA} x_A$ // implied sum on index A
$\mathbf{x}' = R\,\mathbf{x}$ // in vector notation

where R is some rotation matrix ($R^T = R^{-1}$).

Any object which transforms in this way is called a "rank 1 tensor" or, more familiarly, a "vector". A velocity vector **v** would be another example. So **V** is a rank-1 tensor if this is true :

$V'_a = R_{aA} V_A$
$\mathbf{V}' = R\,\mathbf{V}$.

So the position vector in our application is the prototype case of a "vector" (relative to rotations).

To jump ahead, some object Q with four indices is a rank-4 tensor if the following is true

$Q'_{abcd} = R_{aA} R_{bB} R_{cC} R_{dD} Q_{ABCD}$

where now there are four implied sums on A,B,C and D. In this way, one can define how a tensor of any rank transforms (relative to a particular transformation, in this case R).

Our interest will be only in tensors of rank-1 and rank-2. Here is how a rank 2 tensor M is supposed to "transform with respect to underlying transformation R" (R is the thing that rotated the vectors above),

$$M'_{ab} = R_{aA}\, R_{bB}\, M_{AB} \; .$$

Unlike tensors of higher rank, this equation can be written in matrix notation in this way

$$M' = RMR^{T}$$

as is quite easy to verify. This transformation of M into M' is a little reminiscent of our discussion above concerning $D = R^{-1}MR$, and that is why we are off wandering on this tensor tangent at the moment.

Suppose we take our Cartesian axis unit vectors in x-space to be $\hat{\mathbf{e}}^{(i)}$ for i = 1,2..n where $\hat{\mathbf{e}}^{(i)}{}_{j} = \delta_{i,j}$ so for example $\hat{\mathbf{e}}^{(2)} = (0,1,0...0)$. Then one can expand a vector in this obvious manner

$$\mathbf{V} = \Sigma_i \; V_i \, \hat{\mathbf{e}}^{(i)} \qquad \text{where} \qquad V_i = \mathbf{V} \bullet \hat{\mathbf{e}}^{(i)} \; .$$

Now suppose we next define some new rotated basis vectors in this manner

$$\hat{\mathbf{e}}'^{(i)} = R^{-1}\, \hat{\mathbf{e}}^{(i)} \; .$$

When we wrote earlier that $\mathbf{x}' = R\ \mathbf{x}$ and $\mathbf{V}' = R\ \mathbf{V}$, we said that these objects **x** and **V** transformed as vectors with respect to underlying transformation R. But the statement $\hat{\mathbf{e}}'^{(i)} = R^{-1}\, \hat{\mathbf{e}}^{(i)}$ says that $\hat{\mathbf{e}}^{(i)}$ transforms as a vector with respect to a *different* underlying transformation, namely R^{-1}. All is well.

Now here is an interesting claim. The vector **V** can be expanded either on the basis vectors $\hat{\mathbf{e}}^{(i)}$ or on the basis vectors $\hat{\mathbf{e}}'^{(i)}$ as follows,

$$\mathbf{V} = \Sigma_i \; V_i \, \hat{\mathbf{e}}^{(i)}$$

$$\mathbf{V} = \Sigma_i \; V'_i \, \hat{\mathbf{e}}'^{(i)} \qquad \text{where } \hat{\mathbf{e}}'^{(i)} = R^{-1}\, \hat{\mathbf{e}}^{(i)} \text{ and } \mathbf{V}' = R\ \mathbf{V} \; .$$

To show that the second expansion is the same as the first, we will show that (k^{th} component)

$$\Sigma_i \; V_i \, \hat{e}^{(i)}{}_{k} = \Sigma_i \; V'_i \, \hat{e}'^{(i)}{}_{k} \qquad ?$$

$$\text{RHS} = \Sigma_i \; V'_i \, \hat{e}'^{(i)}{}_{k} = \Sigma_i \; [\, R_{ij} V_j] \, [R^{-1}{}_{ks} \, \hat{e}^{(i)}{}_{s}] \qquad \text{// implied sum on j and s}$$

$$= \Sigma_i \; [\, R_{ij} V_j] \, [R^{-1}{}_{ks} \, \delta_{i,s}] = \Sigma_i \; [\, R_{ij} V_j] \, [R^{-1}{}_{ki}] = \Sigma_i (R^{-1}{}_{ki} R_{ij})\, V_j = \delta_{k,j} V_k = V_k$$

$$\text{LHS} = \Sigma_i \; V_i \, \hat{e}^{(i)}{}_{k} = \Sigma_i \; V_i \, \delta_{k,j} = V_k$$

so both sides are the same. So we have two ways to expand the vector **V**.

It can be shown that any tensor of any rank can be expanded in two similar ways. In particular, a rank 2 tensor can be expanded in these two ways,

$$M = \Sigma_{i,j}\ M_{ij}\ \hat{\mathbf{e}}^{(i)}\ \hat{\mathbf{e}}^{(j)T}$$
$$M = \Sigma_{i,j}\ M'_{ij}\ \hat{\mathbf{e}}'^{(i)}\ \hat{\mathbf{e}}^{(j)'T} \qquad \text{where } \hat{\mathbf{e}}'^{(i)} = R^{-1}\ \hat{\mathbf{e}}^{(i)} \text{ and } M'_{ab} = R_{aA}\ R_{bB}\ M_{AB} \quad .$$

This may look a little strange, but the combination $\hat{\mathbf{e}}^{(i)}\ \hat{\mathbf{e}}^{(j)T}$ is just an n x n matrix having a single 1 in a sea of zeros. This 1 is located in row i, column j. For example, assuming n=2,

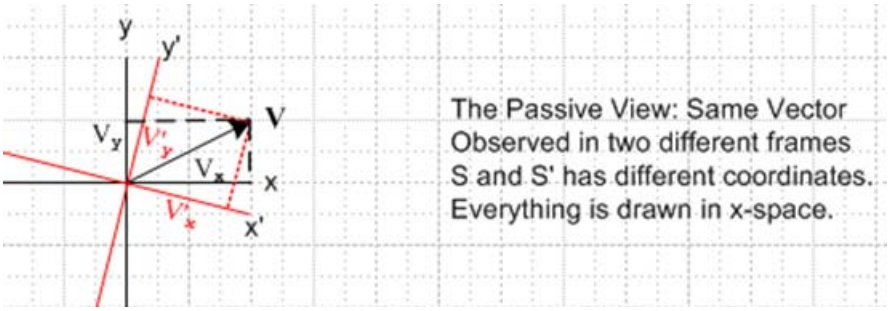
$$\hat{\mathbf{e}}^{(1)}\ \hat{\mathbf{e}}^{(2)T} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} (0\ 1) = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} .$$

The combinations $\hat{\mathbf{e}}'^{(i)}\ \hat{\mathbf{e}}^{(j)'T}$ are also 2x2 matrices, but in general have all elements filled in. This same structure of making a matrix from vectors was encountered at the end of section 4 above where it appears in the definition of the covariance matrix Σ.

Now go back to the pair of vector expansions,

$$\mathbf{V} = \Sigma_i\ V_i\ \hat{\mathbf{e}}^{(i)} \qquad \text{where } V_i = \mathbf{V} \bullet \hat{\mathbf{e}}^{(i)}$$
$$\mathbf{V} = \Sigma_i\ V'_i\ \hat{\mathbf{e}}'^{(i)} \qquad \text{where } V'_i = \mathbf{V} \bullet \hat{\mathbf{e}}'^{(i)}.$$

One can draw a picture to go with these expansions



In this picture

$$\hat{\mathbf{e}}^{(1)} = \hat{\mathbf{x}} \text{ and } \hat{\mathbf{e}}^{(2)} = \hat{\mathbf{y}}$$
$$\hat{\mathbf{e}}'^{(1)} = \hat{\mathbf{x}}' \text{ and } \hat{\mathbf{e}}'^{(2)} = \hat{\mathbf{y}}'$$

and since $\hat{\mathbf{e}}'^{(i)} = R^{-1}\ \hat{\mathbf{e}}^{(i)}$, it appears the $R^{-1} = R_z(-10°)$ since things are rotated the wrong way relative to the usual right hand rule. Then $R = R_z(10°)$.

One would then say that an observer in the black frame of reference views components V_i whereas another observer in the red frame of reference observes components V'_i. These are different sets of numbers, but both sets describe the same vector **V** according to the above expansions.

In the rank-2 tensor case, it is not possible to draw a simple picture like this, but the idea is the same:

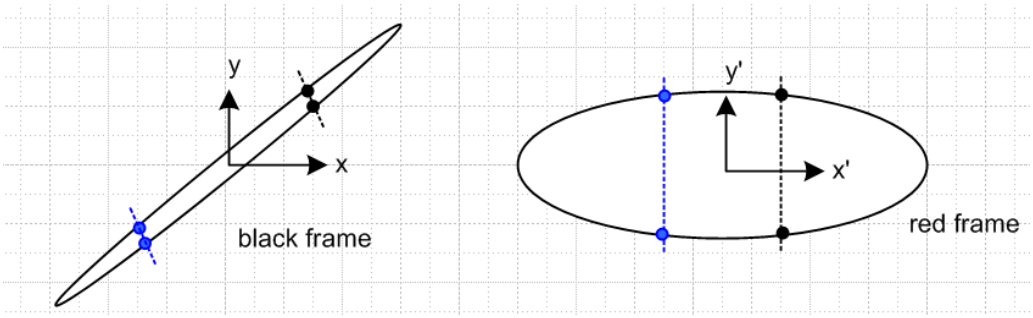
$M = \Sigma_{i,j}\ M_{ij}\ \hat{e}^{(i)}\ \hat{e}^{(j)T}$ where $M_{ij} = \hat{e}^{(i)T} M\ \hat{e}^{(j)}$ (easy to show)

$M = \Sigma_{i,j}\ M'_{ij}\ \hat{e}'^{(i)}\ \hat{e}^{(j),T}$ where $M'_{ij} = \hat{e}'^{(i)T} M\ \hat{e}'^{(j)}$.

The object M is a rank-2 tensor. One would then say that an observer in the black frame of reference views components M_{ij} whereas another observer in the red frame of reference observes components M'_{ij}. We know that if M_{ij} is a symmetric matrix in n dimensions, there is some rotation matrix R which brings M to diagonal form, $M' = RMR^T$. For such a rotation, an observer in the red frame will see M' to be a diagonal matrix, M'_{ij}, whereas the black frame observer sees M_{ij} which is not diagonal.

Example: Imagine a thin elliptical galaxy of stars. The real symmetric covariance tensor Σ_{ij} has some general matrix elements, but there exists a certain "red" frame of reference in which the observer will see Σ'_{ij} as a diagonal matrix. As one would imagine, the basis vectors $\hat{e}'^{(i)}$ are going to line up with the three obvious axes of this thin elliptical galaxy. If $\Sigma' = \text{diag}(\lambda_1,\lambda_2,\lambda_3)$, then λ_1 will be the variance in the "long dimension" of the elliptical galaxy, λ_2 will be the smaller variance going across the galaxy, and λ_3 will be the very small variance in the thin dimension of the galaxy.

This example is perhaps worth a picture or two since it really illustrates why there are off-diagonal covariance matrix elements in the black frame and not in the red frame. Consider then these two views of the galaxy, which we just vaguely represent by a simple means without attempting a real 3D distribution,



In the left picture the galaxy is almost edge-on, we have some "strange" axes x and y, and we show two particular stars as black dots and two others as blue dots. For the two black stars (which are far apart, as the right picture shows), x is positive and y is positive, and this is true for any stars on the upper right, so there is "correlation" in the position and so $\Sigma_{xy} = \Sigma_{12} \neq 0$ and we have an off diagonal element of the Σ matrix (see section 3 above about covariance). The same is true for the blue pair of stars, where x and y are both negative, so there is again correlation. For any star, if x is generally positive, so is y, just due to the position of the galaxy relative to the axes. That is why the situation on the left has off diagonal elements in its Σ.

On the right the axes are chosen as the "principle axes" of the galaxy. For the pair of black stars, x' is positive, but y' could be positive or negative, both possibilities are equally likely in a distribution of stars sense. The same is true for the blue pair of stars (which have negative x'). For this picture, if x' is positive, there is no tendency for y' to also be positive, or to be negative for that matter. That is why $\Sigma'_{x'y'} = \Sigma'_{12} = 0$ for the picture on the right. There is no correlation between the x' and y' coordinates of randomly selected stars in the distribution on the right.

This example is admittedly a bit vague in its wording, but hopefully gives intuition concerning why there are off-diagonal Σ elements in one case and not the other.

7. Positive definite matrices, covariance being an example

A symmetric matrix M_{ij} is positive definite iff $\mathbf{x}^T M\mathbf{x} > 0$ for all vectors $\mathbf{x} \neq \mathbf{0}$. [If $\mathbf{x}^T M\mathbf{x} \geq 0$ then M is called positive *semi*-definite.] A completely equivalent definition of M being positive definite is that, when M is diagonalized into D with eigenvalues λ_i as the diagonal elements, all the λ_i will be positive numbers. Here is a verification of this in one direction. If M is positive definite, then $\mathbf{x}^T M\mathbf{x} > 0$, being true for all vectors, must be true for the eigenvectors $\hat{\mathbf{u}}^{(i)}$ where recall $M\,\hat{\mathbf{u}}^{(i)} = \lambda_i\,\hat{\mathbf{u}}^{(i)}$. Then one has

$$0 < \hat{\mathbf{u}}^{(i)T} M\,\hat{\mathbf{u}}^{(i)} = \hat{\mathbf{u}}^{(i)T}(\lambda_i\,\hat{\mathbf{u}}^{(i)}) = \lambda_i \hat{\mathbf{u}}^{(i)T}\hat{\mathbf{u}}^{(i)} = \lambda_i \hat{\mathbf{u}}^{(i)} \bullet \hat{\mathbf{u}}^{(i)} = \lambda_i \qquad \text{so} \qquad \lambda_i > 0 \;\; i=1..n$$

Proof that covariance matrix is positive definite: Start with the definition as shown earlier.

$$\Sigma = (1/m)\Sigma_{i=1}^{m}\, \mathbf{x}^{(i)}\mathbf{x}^{(i)T}$$

Let **a** be an arbitrary vector and consider,

$$\mathbf{a}^T\Sigma\mathbf{a} = \mathbf{a}^T\,\{(1/m)\Sigma_{i=1}^{m}\, \mathbf{x}^{(i)}\mathbf{x}^{(i)T}\}\mathbf{a} = (1/m)\Sigma_{i=1}^{m}\,(a^T\,\mathbf{x}^{(i)}\mathbf{x}^{(i)T}\mathbf{a})$$

$$= (1/m)\Sigma_{i=1}^{m}\; [\mathbf{x}^{(i)T}\mathbf{a}]^T\,[\mathbf{x}^{(i)T}\mathbf{a}\,] = (1/m)\Sigma_{i=1}^{m}\; [\mathbf{x}^{(i)} \bullet \mathbf{a}]^2 \;\; .$$

Unless the training set is highly contrived, it seems likely that for any vector **a**, there will be at least one term in the sum which is non-zero and such a term will make $\mathbf{a}^T\Sigma\mathbf{a} > 0$. Therefore, the matrix Σ is "positive definite" and its diagonal elements λ_i are then all > 0. A contrived example would be an elliptic galaxy in 3D space which happens to lie exactly in a plane and so has no thickness whatsoever. Then if **a** is normal to that plane, $\mathbf{a}^T\Sigma\mathbf{a} = 0$. Even in this case, Σ would be positive semi-definite and one would have $\lambda_i \geq 0$.

The conclusion is that a covariance matrix is positive definite, and therefore the eigenvalues λ_i of the diagonalized covariance matrix must all be positive. This is a relief, since we know that in the diagonalized covariance matrix the diagonal elements are all variances as observed in a the "red" frame of reference (section 6), and variances must be positive (section 2).

8. Restatement of some results in Ng notation.

In our tensor discussion of Section 6 above, we now want to make these two changes:

(1) $R \rightarrow R^{-1}$ everywhere

(2) $\hat{\mathbf{e}}'^{(i)} \rightarrow \hat{\mathbf{u}}^{(i)}$ // just renaming the rotated (red) system's unit vectors

The $\hat{\mathbf{e}}^{(i)}$ continue to be the orthogonal unit vectors in the original "black frame" feature space. We could have called these $\hat{\mathbf{x}}^{(i)}$ so that for example $\hat{\mathbf{x}}^{(1)}$ is a unit vector along the x_1 axis, but that seems too confusing since $\mathbf{x}^{(i)}$ also refers to a training sample, so we leave them as $\hat{\mathbf{e}}^{(i)}$. In Section 6 we had $\hat{\mathbf{e}}'^{(i)} = R^{-1}\,\hat{\mathbf{e}}^{(i)}$ so with the changes shown above we now have

$$\hat{\mathbf{u}}^{(i)} = R\,\hat{\mathbf{e}}^{(i)}$$

which says that the axes $\hat{\mathbf{u}}^{(i)}$ which describe a (red) frame of reference in which M becomes diagonal M' are obtained from the regular feature space axes by the rotation R. Writing this last equation in components we get

$$\hat{\mathbf{u}}^{(i)}{}_j = [R\,\hat{\mathbf{e}}^{(i)}]_j = R_{jk}\hat{\mathbf{e}}^{(i)}{}_k = R_{jk}\delta_{i,k} = R_{ji}$$

which says that the i^{th} column of matrix R_{ji} is the vector $\hat{\mathbf{u}}^{(i)}$, so

$$R = [\,\hat{\mathbf{u}}^{(1)}, \hat{\mathbf{u}}^{(2)}, \ldots\ldots \hat{\mathbf{u}}^{(n)}\,].$$

Also we now have, applying the above two alterations,

$$M' = R^{-1}MR$$

$$\mathbf{V} = \Sigma_i\ V_i\,\hat{\mathbf{e}}^{(i)}$$
$$\mathbf{V} = \Sigma_i\ V'_i\,\hat{\mathbf{u}}^{(i)} \qquad \text{where } \hat{\mathbf{u}}^{(i)} = R\,\hat{\mathbf{e}}^{(i)} \text{ and } \mathbf{V}' = R^{-1}\,\mathbf{V}$$

$$M = \Sigma_{i,j}\ M_{ij}\,\hat{\mathbf{e}}^{(i)}\,\hat{\mathbf{e}}^{(j)T}$$
$$M = \Sigma_{i,j}\ M'_{ij}\,\hat{\mathbf{u}}^{(i)}\,\hat{\mathbf{u}}^{(j)T} \qquad \text{where } \hat{\mathbf{u}}^{(i)} = R\,\hat{\mathbf{e}}^{(i)} \text{ and } M'_{ab} = R^{-1}{}_{aA}\,R^{-1}{}_{bB}\,M_{AB}$$

As a reminder, we have two expansions for the rank-2 tensor M. In the first expansion, the expansion coefficients (of the $\mathbf{ab}^T$ matrices shown) are M_{ij} which in general is not diagonal. In the second expansion, the coefficients are M'_{ij} and M' is a diagonal matrix with diagonal elements λ_i which are sorted in descending order. This matrix M' was called D in section 5 above.

For the covariance matrix $M = \Sigma$, we now have

$$\Sigma' = R^{-1}\,\Sigma\ R \qquad \Rightarrow$$

$$\Sigma = R\,\Sigma'\,R^{-1} = R\,\Sigma'\,R^T$$

where Σ' is diagonal with descending positive eigenvalues λ_i .

Question: how do we *find* the matrix R which brings Σ into diagonal form Σ', and how do we *find* the diagonal elements of Σ' ?

Answer: We described exactly how to do this above in terms of M and D, but we don't want to do all that work, so instead we use the Octave call

[U, S, V] = svd (Σ),

which in our case is going to be

[R, Σ ', R] = svd (Σ) .

This call then returns on its first argument

R = [$\hat{\mathbf{u}}^{(1)}$, $\hat{\mathbf{u}}^{(2)}$, ... $\hat{\mathbf{u}}^{(n)}$],

and returns on the second argument the diagonal matrix Σ' with diagonal elements sorted in descending order

$\lambda_1 \geq \lambda_2 \geq \lambda_3 ..... \geq \lambda_n$

The third argument will be another copy of R.

The next section explains the svd operation.

9. The Singular Value Decomposition of a matrix

In the most general case and with a certain standard notation, an arbitrary matrix M can be written as the product of three matrices in this manner,

$M = USV^H$

$$\underset{n\,x\,p}{M} = \underset{n\,x\,n}{U}\ \underset{n\,x\,p}{S}\ \underset{p\,x\,p}{V^H}$$

where we show the row x column sizes of the matrices. In general the elements of the matrices are complex numbers, and we have used ^{H} to indicate the "Hermitian conjugate" , as in

$V^H = (V^T)^*$ or $V^H_{ij} = V_{ji}^*$

where * means complex conjugation. Sadly, there are several notations for complex conjugate, Hermitian conjugate, and transpose, and they conflict somewhat with one another (we use the first notation listed in each case)

complex conjugate	use *; use overbar	
Hermitian conjugate	use ^{H} ; use † ; use * ; use $^+$	// aka "adjoint"
transpose	use ^{T}; use overtwiddle; use ' (Octave)	

In this decomposition, the square matrices U and V are both unitary, which means $U^H = U^{-1}$ and

$V^H = V^{-1}$. The non-square matrix S is all zeros except for some real numbers on its "diagonal" which starts in the upper left corner. These numbers are real and are called "**the singular values**" of matrix M and they are conventionally listed in monotonic decreasing order.

If we write the square matrices U and V matrices in terms of their columns in this manner

$$U = [\hat{\mathbf{u}}^{(1)}, \hat{\mathbf{u}}^{(2)}, ... \hat{\mathbf{u}}^{(n)}]$$

$$V = [\hat{\mathbf{v}}^{(1)}, \hat{\mathbf{v}}^{(2)}, ... \hat{\mathbf{v}}^{(n)}]$$

then the $\hat{\mathbf{u}}^{(i)}$ are eigenvectors of MM^H and the $\hat{\mathbf{v}}^{(i)}$ are eigenvectors of M^HM. Notice that both these matrices MM^H and M^HM are Hermitian ($A=A^H$) and therefore *have* eigenvectors. For real numbers only, the condition is that $A = A^T$ (symmetric) for a matrix to have eigenvectors. The claim is that the non-zero singular values (on the diagonal of S) are the square roots of the non-zero eigenvalues of MM^H and M^HM.

Our application of the SVD decomposition is much simpler than this general case. First of all, we have only real numbers, so the unitary matrices U and V ($U^HU=1$ and $V^HV=1$) become real orthogonal matrices ($U^TU=1$ and $V^TV=1$). Secondly, dimension p = n, so *all* matrices are square n x n. In this case, then, the SVD says

$M = U\ S\ V^T$ where U and V are real orthogonal and S is diagonal.

In Ng's application of SVD, we identify U = V = R where R is real orthogonal so $R^{-1} = R^T$, and we identify S with M', our diagonal matrix, so the decomposition is then

$M = U\ S\ V^T = R\ M'\ R^T$ since $M' = R^{-1}M\ R$ (see section 8) .

Since M is real symmetric, it has some eigenvectors $\hat{\mathbf{u}}^{(i)}$ and eigenvalues λ_i. Then we can verify the claim made above that the diagonal elements of S should be the square roots of the eigenvalues of MM^T and M^TM. Of course here these two matrices are the same ($M = M^T$) and are equal to M^2 so:

$$MM^T\hat{\mathbf{u}}^{(i)} = M^TM\ \hat{\mathbf{u}}^{(i)} = M^2\ \hat{\mathbf{u}}^{(i)} = M\ \lambda_i\hat{\mathbf{u}}^{(i)} = \lambda_i\ M\hat{\mathbf{u}}^{(i)} = \lambda_i\ \lambda_i\hat{\mathbf{u}}^{(i)} = \lambda_i^2\ \hat{\mathbf{u}}^{(i)} .$$

If M is a positive definite matrix, then the descending elements of S = M' are the eigenvalues of M,

$$\lambda_1 \geq \lambda_2 ..... \geq \lambda_n > 0$$

and the eigenvectors which are the columns of U = R are the corresponding eigenvectors.

Main Point: We are trying to find the rotation matrix R which diagonalizes M into M'

$$M' = R^TM\ R$$

such that the diagonal elements of M' are in descending order. If we call the SVD routine with M as argument and ask for three returned matrices,

[U, S, V] = svd (M) meaning $M = U\,S\,V^T = R\;M'\,R^T$

then U is our desired matrix R, and S is our desired matrix M'.

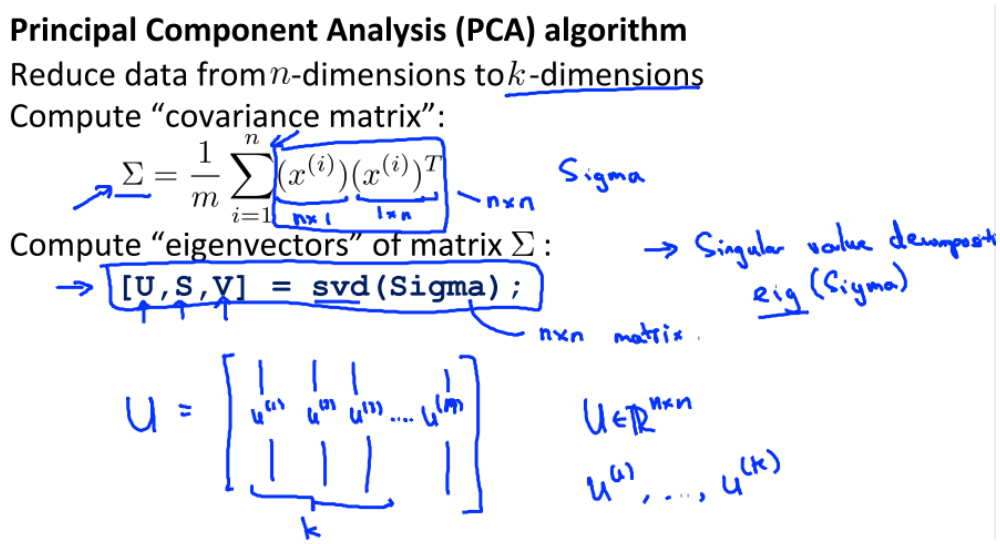
For the matrix M = Σ, the covariance matrix, we have

[U, S, V] = svd (Σ) meaning $\Sigma = U\,\Sigma'\,V^T = R\;\Sigma'\,R^T$

which can then be written

[R, Σ', R] = svd (Σ)

All of the notes in the above nine sections are compressed into this one Ng lecture slide:



10. Data Reduction: throwing out variance

In our ongoing 3D super-thin elliptic galaxy example, it does seem intuitive that most of the useful information regarding the distribution of the stars is contained in the plane which aligns with the galaxy, and if you projected all the stars of the actual galaxy onto this plane, you would not lose much in a statistical description of the galaxy.

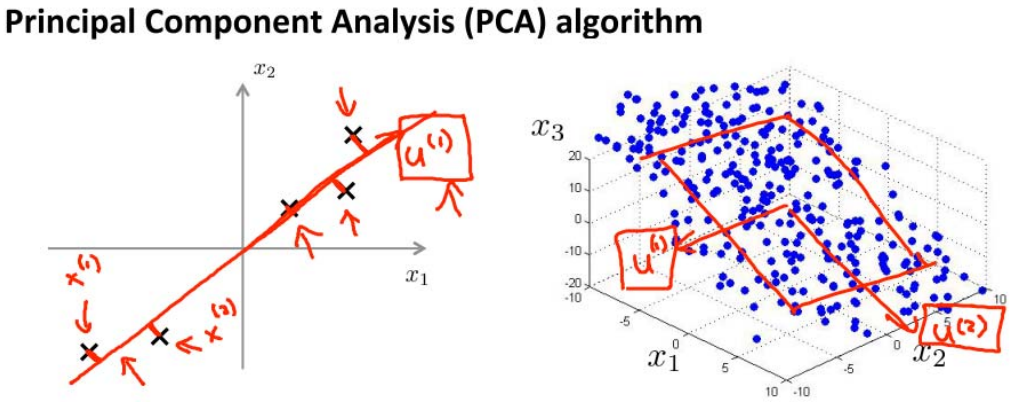
In the rotated frame where the covariance tensor is diagonal, $\Sigma'_{ij} = \delta_{i,j}\lambda_i$ with the λ_i in descending order, each λ_i is the variance of the distribution of training sample points in the i^{th} direction $\hat{\mathbf{u}}^{(i)}$. So the dimensions of largest variance are listed first and the smallest come at the end. Perhaps for n = 4,

$$\Sigma' = \begin{pmatrix} 12 & 0 & 0 & 0 \\ 0 & 6 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0.3 \end{pmatrix} .$$

If one were to "flatten" (squash flat, project) the distribution in the last two principal directions, causing 0 variance in these two directions, one would have

$$\Sigma'_{\mathtt{approx}} = \begin{pmatrix} 12 & 0 & 0 & 0 \\ 0 & 6 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

This is the idea suggested by these Ng graphics for n = 2 and n = 3,



Without further ado, we can compute how much variance is "lost" by doing such a compression. Suppose we decide to keep only first k eigenvalues λ_i and throw out (set to 0) the remaining ones. Since these diagonal elements (being diagonal elements of a covariance matrix) are in fact the variances in the rotated space, the total variance summed over all directions of our training samples is $\Sigma_{i=1}^{n} \lambda_i$. Thus

fraction of variance retained = $(\Sigma_{i=1}^{k} \lambda_i) / (\Sigma_{i=1}^{n} \lambda_i.)$

fraction of variance thrown out = $(\Sigma_{i=k+1}^{n} \lambda_i) / (\Sigma_{i=1}^{n} \lambda_i.)$

= 1 - $(\Sigma_{i=1}^{k} \lambda_i) / (\Sigma_{i=1}^{n} \lambda_i.)$

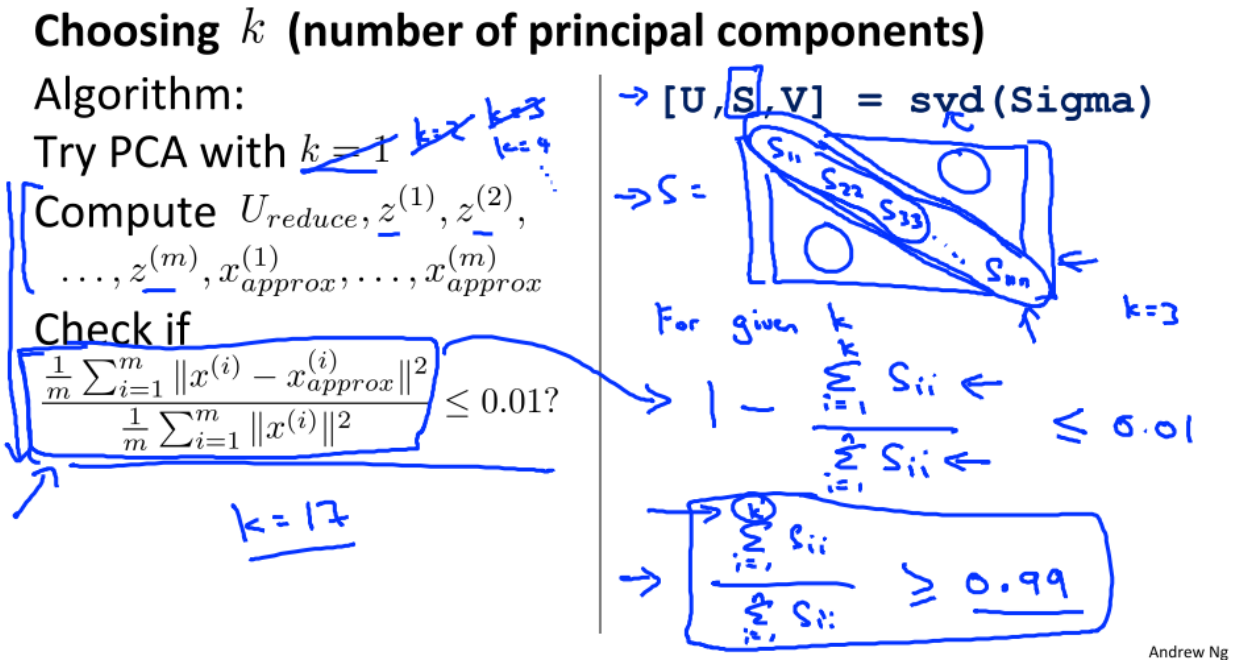
In Prof. Ng's discussion, the diagonal covariance matrix Σ' is called S and then we have

fraction of variance retained = $(\Sigma_{i=1}^{k} S_{ii}) / (\Sigma_{i=1}^{n} S_{ii}.)$

fraction of variance thrown out = $(\Sigma_{i=k+1}^{n} S_{ii}) / (\Sigma_{i=1}^{n} S_{ii}.)$

= 1 - $(\Sigma_{i=1}^{k} S_{ii}) / (\Sigma_{i=1}^{n} S_{ii}.)$

which then agrees with the right side of Prof. Ng's slide (we will do the left side in the next section)



He suggests that if one throws out ~1% or less of the total variance, the compression won't affect things much. Stated another way in his next slide (bottom sum should go to n)

Pick smallest value of k for which

$$\frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{m} S_{ii}} \geq 0.99$$

k=100

(99% of variance retained)

11. Data Reduction: doing the actual data compression (PCA)

(Note: observation reference frames called S and S' here have no relation to S_{ij} mentioned above.)

For this little section, we regard the original training samples as $\mathbf{x}^{(i)}$ viewed from reference frame S. When viewed from the rotated frame S' in which covariance is diagonal, the samples are $\mathbf{x}'^{(i)}$. The unit vectors in frame of reference S are the $\hat{\mathbf{e}}^{(i)}$, and the unit vectors in the rotated frame S' are the $\hat{\mathbf{e}}'^{(i)}$ which we agreed to rename $\hat{\mathbf{u}}^{(i)}$. The mapping between the $\mathbf{x}^{(i)}$ and the $\mathbf{x}'^{(i)}$ is given by (see section 8)

$\mathbf{x}'^{(i)} = R^{-1}\,\mathbf{x}^{(i)}$ for all the training samples i = 1,2...m

$\hat{\mathbf{u}}^{(j)} = R\,\hat{\mathbf{e}}^{(j)}$ for all the axes j = 1,2...n .

The plan is that in frame S', we are going to just throw out (set to 0) the last n-k components of every training vector $\mathbf{x}'^{(i)}$. Since the original samples were pre-processed to have zero mean in all directions, which is to say $\boldsymbol{\mu} = 0$ in frame S, this fact $\boldsymbol{\mu} = 0$ is also true in the rotated frame S'. Thus, setting all the last components of $\mathbf{x}'^{(i)}$ to zero is the same as squashing flat those last n-k dimensions of the distribution.

For example, we might have

$$\mathbf{x}'^{(i)T} = (x'^{(i)}_1, x'^{(i)}_2, x'^{(i)}_3, x'^{(i)}_4, x'^{(i)}_5, x'^{(i)}_6)$$

and if k = 4, then after compression we have

$$\mathbf{x}'^{(i)T}_{approx} = (x'^{(i)}_1, x'^{(i)}_2, x'^{(i)}_3, x'^{(i)}_4, 0, 0) \quad .$$

To find the location of the $\mathbf{x}'^{(i)}_{approx}$ back in frame S, we have (R = U = matrix with n rows)

$$\mathbf{x}^{(i)}_{approx} = R\, \mathbf{x}'^{(i)}_{approx}$$

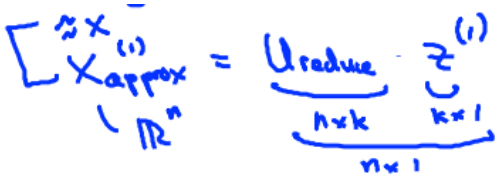
$$= (\hat{\mathbf{u}}^{(1)}, \hat{\mathbf{u}}^{(2)}, \hat{\mathbf{u}}^{(3)}, \hat{\mathbf{u}}^{(4)}, \hat{\mathbf{u}}^{(5)}, \hat{\mathbf{u}}^{(6)}) \begin{pmatrix} x'^{(i)}_1 \\ x'^{(i)}_2 \\ x'^{(i)}_3 \\ x'^{(i)}_4 \\ 0 \\ 0 \end{pmatrix} = U\, \mathbf{x}'^{(i)}_{approx} \quad .$$

Note that this does NOT say the last components of $\mathbf{x}^{(i)}_{approx}$ are zero (frame of reference S). The points here called $\mathbf{x}^{(i)}_{approx}$ are the original $\mathbf{x}^{(i)}$ which have been projected onto a lower dimensional flat surface in $\mathcal{R}^n$.

In the above product of a matrix times a vector, since the last two elements of $\mathbf{x}'^{(i)}_{approx}$ are 0, the columns labeled $\hat{\mathbf{u}}^{(5)}$ and $\hat{\mathbf{u}}^{(6)}$ play no role and the above product is unchanged if we write it this way,

$$\mathbf{x}^{(i)}_{approx} = (\hat{\mathbf{u}}^{(1)}, \hat{\mathbf{u}}^{(2)}, \hat{\mathbf{u}}^{(3)}, \hat{\mathbf{u}}^{(4)}) \begin{pmatrix} x'^{(i)}_1 \\ x'^{(i)}_2 \\ x'^{(i)}_3 \\ x'^{(i)}_4 \end{pmatrix} = U_{reduce}\, \mathbf{z}'^{(i)}$$

where now U_{reduce} has n = 6 rows, but only k =4 columns, so it is an n x k matrix. And we have given a new name $\mathbf{z}'^{(i)}$ to the truncated training samples in S' space, where $\mathbf{z}'^{(i)}$ has only k = 4 components. Compare the above equation to this piece of an Ng slide



We have put a prime on $\mathbf{z}'^{(i)}$ as a reminder that $\mathbf{z}'^{(i)}$ is just the truncated $\mathbf{x}'^{(i)}$

$$\mathbf{z}'^{(i)} = \begin{pmatrix} x'^{(i)}_1 \\ x'^{(i)}_2 \\ x'^{(i)}_3 \\ x'^{(i)}_4 \end{pmatrix} = \text{truncated } \mathbf{x}'^{(i)} = \text{top of } \mathbf{x}'^{(i)}_{\text{approx}} = \text{top of } \begin{pmatrix} x'^{(i)}_1 \\ x'^{(i)}_2 \\ x'^{(i)}_3 \\ x'^{(i)}_4 \\ 0 \\ 0 \end{pmatrix}.$$

The components of $\mathbf{z}'^{(i)}$ are numbers observed in the rotate S' frame. Prof. Ng uses no prime and just calls this thing $\mathbf{z}^{(i)}$, so we will now drop the prime as well.

Going the other direction we have

$$\mathbf{x}'^{(i)} = R^{-1}\,\mathbf{x}^{(i)} = R^{T}\,\mathbf{x}^{(i)}$$

or

$$\begin{pmatrix} x'^{(i)}_1 \\ x'^{(i)}_2 \\ x'^{(i)}_3 \\ x'^{(i)}_4 \\ x'^{(i)}_5 \\ x'^{(i)}_6 \end{pmatrix} = \left(\hat{\mathbf{u}}^{(1)}, \hat{\mathbf{u}}^{(2)}, \hat{\mathbf{u}}^{(3)}, \hat{\mathbf{u}}^{(4)}, \hat{\mathbf{u}}^{(5)}, \hat{\mathbf{u}}^{(6)} \right)^{T} \begin{pmatrix} x^{(i)}_1 \\ x^{(i)}_2 \\ x^{(i)}_3 \\ x^{(i)}_4 \\ x^{(i)}_5 \\ x^{(i)}_6 \end{pmatrix} = \begin{pmatrix} \hat{\mathbf{u}}^{(1)T} \\ \hat{\mathbf{u}}^{(2)T} \\ \hat{\mathbf{u}}^{(3)T} \\ \hat{\mathbf{u}}^{(4)T} \\ \hat{\mathbf{u}}^{(5)T} \\ \hat{\mathbf{u}}^{(6)T} \end{pmatrix} \begin{pmatrix} x^{(i)}_1 \\ x^{(i)}_2 \\ x^{(i)}_3 \\ x^{(i)}_4 \\ x^{(i)}_5 \\ x^{(i)}_6 \end{pmatrix}$$

The top k = 4 components on the left are the desired $\mathbf{z}^{(i)}$ vector and we don't care about the lower two components because we are compressing them away. So for the information we want, the above equation can be replaced by

$$\mathbf{z}^{(i)} = \begin{pmatrix} x'^{(i)}_1 \\ x'^{(i)}_2 \\ x'^{(i)}_3 \\ x'^{(i)}_4 \end{pmatrix} = \begin{pmatrix} \hat{\mathbf{u}}^{(1)T} \\ \hat{\mathbf{u}}^{(2)T} \\ \hat{\mathbf{u}}^{(3)T} \\ \hat{\mathbf{u}}^{(4)T} \end{pmatrix} \begin{pmatrix} x^{(i)}_1 \\ x^{(i)}_2 \\ x^{(i)}_3 \\ x^{(i)}_4 \\ x^{(i)}_5 \\ x^{(i)}_6 \end{pmatrix}$$

where the matrix has k = 4 rows and n=6 columns. The above vector equation can then be written

$$\mathbf{z}^{(i)} = (U_{\text{reduce}})^{T}\,\mathbf{x}^{(i)}$$

which agrees with this Ng clip

$$\rightarrow\ z = U^{T}_{reduce} x$$

So now the plan is this:

(1) find the matrix R = U which diagonalizes the covariance matrix Σ and examine diagonal Σ'

(2) decide how many of the coordinates to keep (k) to keep variance loss at less than perhaps 1%
(3) process all the $\mathbf{x}^{(i)}$ as shown in the last equation above to get the points $\mathbf{z}^{(i)}$
(4) take the new compressed training points $\mathbf{z}^{(i)}$ and use them in place of the original training points $\mathbf{x}^{(i)}$ and replace this data fitting /classification problem in S space

$(\mathbf{x}^{(i)}, y^{(i)})$ $\mathbf{x}^{(i)}$ each have n components m training samples

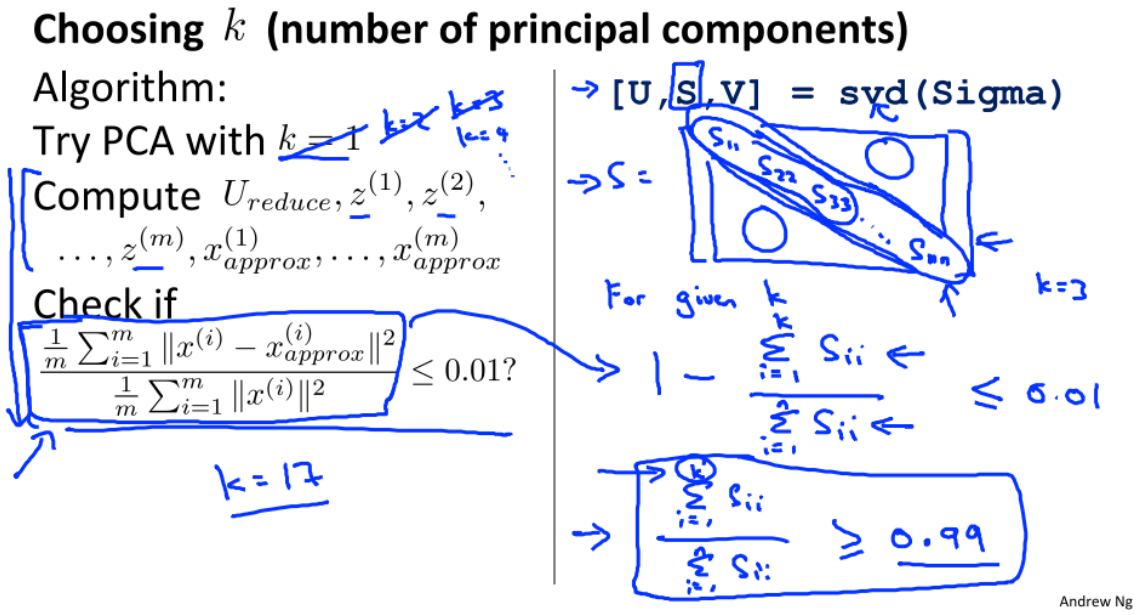
with this new and hopefully faster-to-compute data fitting/classification problem in S' space

$(\mathbf{z}^{(i)}, y^{(i)})$ $\mathbf{z}^{(i)}$ each have k components , $k < n$ m training samples

This data reduction method is called the **Principle Component Algorithm** or **PCA**.

12. Alternate expression for the fraction of data thrown out doing PCA

Here is one of Prof. Ng's slides,



In section 10 above we have already derived the expressions on the right for the fraction of variance retained and thrown out. Here we want to derive the expression shown on the left side of the slide,

fraction of variance thrown out = $(1/m)\,\Sigma_{i=1}^{m} \| \mathbf{x}^{(i)} - \mathbf{x}^{(i)}_{approx} \|^2$ / $(1/m)\,\Sigma_{i=1}^{m} \| \mathbf{x}^{(i)} \|^2$.

The denominator can be written

$(1/m)\,\Sigma_{i=1}^{m} \| \mathbf{x}^{(i)} \|^2 = (1/m)\,\Sigma_{i=1}^{m} \mathbf{x}^{(i)} \bullet \mathbf{x}^{(i)}$

$= (1/m)\,\Sigma_{i=1}^{m}\,\Sigma_{s=1}^{n} (x^{(i)}_s)^2$

$= \Sigma_{s=1}^{n} \{ (1/m) \Sigma_{i=1}^{m} (x^{(i)}_s)^2 \}$

$= \Sigma_{s=1}^{n} \{ \Sigma_{ss} \}$ // beware overloaded symbol Σ

$= \mathrm{tr}(\Sigma)$

where Σ is the covariance matrix given at the end of section 4,

$$\Sigma_{ab} \equiv (1/m) \Sigma_{i=1}^{m} x^{(i)}_a \, x^{(i)}_b \ .$$

Since the trace operation is invariant under $\Sigma' = R^{-1}\Sigma R$ (section 5), we can continue the above steps,

$= \mathrm{tr}(\Sigma') = \Sigma_{s=1}^{n} \{ \Sigma'_{ss} \} = \Sigma_{s=1}^{n} \lambda_s$,

and so we have evaluated the denominator of our claimed fraction to be

$$(1/m) \Sigma_{i=1}^{m} \| \mathbf{x}^{(i)} \|^2 = \Sigma_{s=1}^{n} \lambda_s \ .$$

Next, we shall evaluate the numerator. Now the distance between two points as indicated by $\| \ \|$ is invariant under rotations, so we can evaluate the numerator in our S' rotated frame of reference so that

$$(1/m) \Sigma_{i=1}^{m} \| \mathbf{x}^{(i)} - \mathbf{x}^{(i)}_{approx} \|^2 = (1/m) \Sigma_{i=1}^{m} \| \mathbf{x}'^{(i)} - \mathbf{x}'^{(i)}_{approx} \|^2$$

$$= (1/m) \Sigma_{i=1}^{m} \Sigma_{s=1}^{n} (x'^{(i)}_s - x'^{(i)}_{approx,s})^2 \ .$$

As shown in the previous section, $\mathbf{x}'^{(i)}$ and $\mathbf{x}'^{(i)}_{approx}$ are the same in the first k components. Thus

$$= (1/m) \Sigma_{i=1}^{m} \Sigma_{s=k+1}^{n} (x'^{(i)}_s - x'^{(i)}_{approx,s})^2 \ .$$

But in the last n-k components, $\mathbf{x}'^{(i)}_{approx}$ vanishes, so we have

$= (1/m) \Sigma_{i=1}^{m} \Sigma_{s=k+1}^{n} (x'^{(i)}_s)^2$

$= \Sigma_{s=k+1}^{n} \{ 1/m) \Sigma_{i=1}^{m} (\mathbf{x}'^{(i)}_s)^2 \}$

$= \Sigma_{s=k+1}^{n} \{ \Sigma'_{ss} \}$

$= \Sigma_{s=k+1}^{n} \lambda_s$.

Thus we have shown these facts:

$$(1/m) \Sigma_{i=1}^{m} \| \mathbf{x}^{(i)} \|^2 = \Sigma_{s=1}^{n} \lambda_s$$

$$(1/m)\,\Sigma_{i=1}^{m} \| \mathbf{x}^{(i)} - \mathbf{x}^{(i)}_{approx} \|^2 = \Sigma_{s=k+1}^{n} \lambda_s$$

Thus, the claim being made is that

$$\text{fraction thrown out} = \Sigma_{s=k+1}^{n} \lambda_s \;/\; \Sigma_{s=1}^{n} \lambda_s$$

and this agrees with what we already found out in section 10.

Comment: It seems reasonable to use PCA when one has perhaps many unimportant dimensions all having small variance compared with the major dimensions of the problem. Those low-variance dimensions aren't providing much useful information in terms of doing a fit or a classification. If their presence slows down the calculation significantly, that is when they should be tossed overboard.

F. Notes on the Multivariate Gaussian

1. The multivariate Gaussian idea

This section makes use of ideas from Section E.1 through E.8, so those items won't be rederived here.

Consider a pdf of this form (N is a normalization constant we will compute in the next section),

$$p(\mathbf{x}) = N \exp(- \mathbf{x}^T \Sigma^{-1} \mathbf{x}/2).$$

The quantity $\mathbf{x}^T \Sigma^{-1} \mathbf{x}$ is exactly the "quadratic form" type object mentioned above in the comment in Section E.5. In x-space ("black" coordinate system x_i with original feature space axes, Section E.6), the equation $\mathbf{x}^T \Sigma^{-1} \mathbf{x} = k^2$ (a constant) describes some skiwampous rotated ellipsoid centered at the origin, a fact we shall prove in a moment. As k^2 takes a set of ever-increasing values, the ellipsoids comprise a family of ever-larger concentric ellipsoids in feature space on each of which $p(\mathbf{x})$ takes some constant but ever-*de*creasing value, namely, $\exp(-k^2/2)$.

Assume that the x-space unit vectors are $\hat{\mathbf{e}}^{(i)}$. As discussed in Section E.8, we know there is some rotation R such that the following facts are true,

$\hat{\mathbf{u}}^{(i)} = R\, \hat{\mathbf{e}}^{(i)}$, i = 1,2...n // unit vectors in a "red" rotated frame of reference (x'-space)

$\Sigma' = R^{-1} \Sigma R$ // matrix Σ' is diagonal in this rotated frame (x'-space)

$\mathbf{x}' = R^{-1}\, \mathbf{x}$. // vector $\mathbf{x}$ observed in rotated frame is $\mathbf{x}'$

The last line says that the vector $\mathbf{x}$ in the original feature space with components x_i has components

$$x'_i = \Sigma_j (R^{-1})_{ij} x_j$$

in the rotated frame of reference. The middle line says that the special rotation R brings Σ to the diagonal form Σ' so that $\Sigma'_{ij} = \delta_{i,j}\, \lambda_i$ where the λ_i are the eigenvalues of both matrices Σ and Σ'.

The matrix Σ'^{-1} must have this simple form

$$\Sigma'^{-1}_{ij} = \delta_{i,j}\, (1/\lambda_i) \qquad \text{whereas:} \qquad \Sigma'_{ij} = \delta_{i,j}\, \lambda_i$$

proof: $(\Sigma'^{-1}\Sigma')_{ik} = \Sigma_j\ \Sigma'^{-1}_{ij}\, \Sigma'_{jk} = \Sigma_j\ \delta_{i,j}\, (1/\lambda_i\ )\, \delta_{j,k}\, \lambda_k = \delta_{i,k}\, (1/\lambda_i\ )\ \lambda_k = \delta_{i,k}$.

This result is immediately obvious if you just draw a picture of the matrix multiplication $\Sigma'^{-1}\, \Sigma' = 1$. So Σ'^{-1} is diagonal and the diagonal elements are the *inverses* of the eigenvalues λ_i.

Now consider

$$\Sigma'^{-1} = (R^{-1} \Sigma R)^{-1} = R^{-1} \Sigma^{-1} R \qquad \text{compare to:} \qquad \Sigma' = R^{-1} \Sigma R$$

Therefore, the same rotation R that brings Σ to diagonal form Σ', also brings Σ^{-1} to diagonal form Σ'^{-1}.

Now, using the facts assembled above we find that

$$\mathbf{x}^T\Sigma^{-1}\mathbf{x} = (R\mathbf{x}')^T(R\Sigma'^{-1}R^{-1})(R\mathbf{x}') = \mathbf{x}'^T(R^TR)\Sigma'^{-1}(R^{-1}R)\mathbf{x}' = \mathbf{x}'^T\Sigma'^{-1}\mathbf{x}'$$

where, once again, $\mathbf{x}'$ is the vector $\mathbf{x}$ observed in a rotated reference frame which has axes $\hat{\mathbf{u}}^{(i)}$. Thus, in this "red" frame of reference (and coordinate system), we have

$$p(\mathbf{x}) = N \exp(- \mathbf{x}'^T\Sigma'^{-1}\mathbf{x}'/2) \equiv P(\mathbf{x}') \ .$$

But *in* this rotated "red" reference frame where Σ' is diagonal we have (for our constant k^2 above)

$$\mathbf{x}'^T\Sigma'^{-1}\mathbf{x}' = \Sigma_{i=1}^{n}\lambda_i^{-1}x'^2_i = k^2 \quad \Rightarrow \quad \Sigma_{i=1}^{n}[\,x'^2_i/A_i^2] = 1 \quad \text{where } A_i = k\sqrt{\lambda_i}$$

or

$$\frac{x'^2_1}{A_1^2} + \frac{x'^2_2}{A_2^2} + ... \frac{x'^2_n}{A_n^2} = 1 \qquad A_i = \text{the } i^{th} \text{ ellipsoid semi-major axes} \ .$$

This is an axis-aligned ellipsoidal surface in n dimensions centered at the origin, a generalization of an ellipse in 2 dimensions. Since it is totally clear that this is an ellipsoid in x'-space (red), it must also be an ellipsoid in x-space (black), since the red and black frames of reference are related by the rotation R. This then is our proof promised above.

Notice that the semi-major axes of the aligned ellipsoid $\mathbf{x}'^T\Sigma'^{-1}\mathbf{x}' = k^2$ is k times the square root of the eigenvalue λ_i of the matrix Σ. But these eigenvalues of the covariance matrix Σ are the variances in the various directions, so we conclude that the semi-major axes A_i of the above ellipsoid labeled by k are in fact k times the standard deviations which are the square roots of the variances!

In this special red "principal axis" reference frame in which Σ' and Σ'^{-1} are diagonal, we can write

$$P(\mathbf{x}') = N \exp(- \mathbf{x}'^T\Sigma'^{-1}\mathbf{x}'/2) = N \exp(- \Sigma_{i=1}^{n}\lambda_i^{-1}x'^2_i/2) = N \exp(- \Sigma_{i=1}^{n}x'^2_i/2\lambda_i)$$

$$= N \Pi_{i=1}^{n} \exp(- x'^2_i/2\lambda_i) = N \exp(- x'^2_1/2\lambda_1) \exp(- x'^2_2/2\lambda_2).... \exp(- x'^2_n/2\lambda_n)$$

which is a product of Gaussians in the various dimensions, and in each dimension, the standard deviation is in fact $\sigma_i^2 = \lambda_i$, as just noted in the previous paragraph. That is to say, in the normal way the Gaussian is written, one has $G = \exp(-x^2/2\sigma^2)$, so we identify standard deviation σ with $\sqrt{\lambda_i}$.

We know from Section E.3 that this factored form for P means there is no correlation between any pair of coordinates, and that of course agrees with the fact that Σ' is diagonal. Recall that the off-diagonal elements of the covariance matrix Σ in general are correlations, while the diagonal elements are variances.

We showed above that $\mathbf{x}'^T\Sigma'^{-1}\mathbf{x}' = \mathbf{x}^T\Sigma^{-1}\mathbf{x}$. A number which is invariant under a transformation is called a "scalar" so $\mathbf{x}^T\Sigma^{-1}\mathbf{x}$ is a scalar with respect to the rotation R^{-1}. Another scalar is $\mathbf{x}^T\mathbf{x}$, whereas $\mathbf{x}$ itself transforms as a vector $\mathbf{x}' = R^{-1}\mathbf{x}$.

Example**:** Once again we consider our thin elliptical galaxy situated at some skewed angles in feature space where the stars represent training features $x^{(i)}$. We assume that this galaxy is centered on our feature space origin (zero mean in all dimensions, $\boldsymbol{\mu} = 0$). If we hop into a reference frame (x'-space) that

is aligned with the galaxy's principal axes, a reasonable model for the distribution of stars in the galaxy might be the function

$$P(\mathbf{x}') = N \exp(- x'_1{}^2/2\lambda_1) \exp(- x'_2{}^2/2\lambda_2)\exp(- x'_3{}^2/2\lambda_3) ,$$

where the λ_i are the variances of the galaxy distribution in the three directions. In the original feature space with coordinates x_i, this same distribution is written

$$P(\mathbf{x}') = N \exp(- \mathbf{x}'^T\Sigma'^{-1}\mathbf{x}'/2) = N \exp(- \mathbf{x}^T\Sigma^{-1}\mathbf{x}/2) = p(\mathbf{x})$$

where we make use of the scalar nature of $\mathbf{x}^T\Sigma^{-1}\mathbf{x}$. Of course in this x-space, the exponential can no longer be factored into a product of exponentials, and this corresponds to the fact that in x-space, Σ is not diagonal and there will be correlations.

Conclusion: If we have some distribution of training points $x^{(i)}$ in feature space and we want to attempt "a Gaussian fit", the right thing to do is compute the covariance matrix

$$\Sigma = \Sigma_{i=1}^m \mathbf{x}^{(i)}\mathbf{x}^{(i)T}$$

and then use as the Gaussian fit the multivariate Gaussian given by

$$p(\mathbf{x}) = N \exp(- \mathbf{x}^T\Sigma^{-1}\mathbf{x}/2) .$$

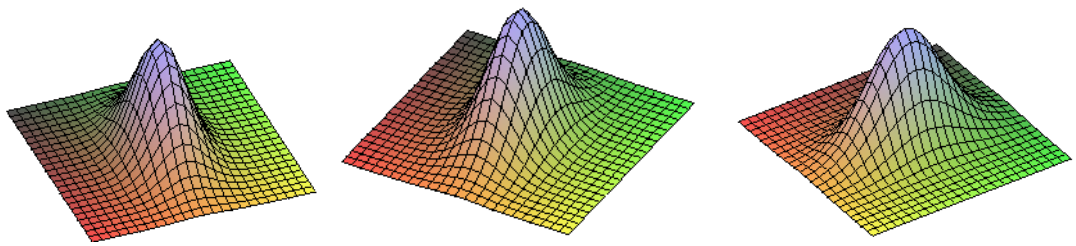
Here we have assumed that the training points $x^{(i)}$ have been pre-processed so as to have zero mean. If this is not the case, we just replace $\mathbf{x}$ by $\mathbf{x}-\boldsymbol{\mu}$ in everything above:

$$\Sigma = \Sigma_{i=1}^m [\mathbf{x}^{(i)}-\boldsymbol{\mu}] [\mathbf{x}^{(i)}-\boldsymbol{\mu}]^T \qquad \text{// covariance matrix}$$

$$p(\mathbf{x}) = N\exp(- [\mathbf{x}-\boldsymbol{\mu}]^T\Sigma^{-1}[\mathbf{x}-\boldsymbol{\mu}]/2) \qquad \text{// multivariate Gaussian}$$

$$\boldsymbol{\mu} = (1/m) \Sigma_{i=1}^m \mathbf{x}^{(i)} \qquad \text{// mean}$$

This p(x) is then the rotated Gaussian Prof. Ng draws several times in his slides, and the test for an anomalous new test point $\mathbf{x}$ would be $p(\mathbf{x}) < \varepsilon$ for some ε that has perhaps been vindicated by some practice data which includes some known anomalous outlier points. The condition $p(\mathbf{x}) < \varepsilon$ graphically means that the point x is way low in the "skirt" of the multivariate Gaussian, below our threshold of comfort. Here is a Maple plot plot3d(exp(-x^2 - 3*y^2), x=-2..2, y=-2..2); from several views,

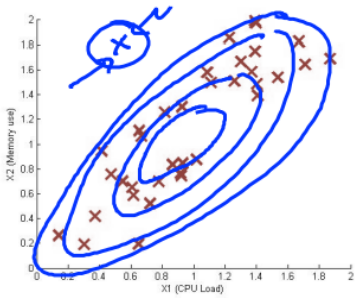


Anomaly detection with the multivariate Gaussian

1. Fit model $p(x)$ by setting

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$$

$$\Sigma = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)} - \mu)(x^{(i)} - \mu)^T$$



2. Given a new example x, compute

$$p(x) = \frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|^{\frac{1}{2}}}\exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$$

Flag an anomaly if $p(x) < \varepsilon$

Andrew Ng

2. Normalization

Here we derive the normalization factor N shown above. The condition is

$$\int d^n x\, p(\mathbf{x}) = 1$$

where we integrate over all of $\mathbb{R}^n$. Thus

$$N\int d^n x \exp(-[\mathbf{x}-\boldsymbol{\mu}]^T\Sigma^{-1}[\mathbf{x}-\boldsymbol{\mu}]/2) = 1$$

or

$$1/N = \int d^n x \exp(-[\mathbf{x}-\boldsymbol{\mu}]^T\Sigma^{-1}[\mathbf{x}-\boldsymbol{\mu}]/2)\,.$$

By changing from integration variables $\mathbf{x}$ to $\mathbf{y} = \mathbf{x}-\boldsymbol{\mu}$ and noting that $d^n y = d^n x$ and the integration endpoints stay at $-\infty,\infty$ for all variables, one can simplify the above to be ($\mathbf{y} \rightarrow \mathbf{x}$ when done)

$$1/N = \int d^n x \exp(-\mathbf{x}^T\Sigma^{-1}\mathbf{x}/2)\,.$$

Next, since $\mathbf{x}^T\Sigma^{-1}\mathbf{x}$ is a scalar, we might as well evaluate the integral in the principal axis frame where we know the exponential is going to factor as shown above

$$1/N = \int d^n x \exp(-\mathbf{x}'^T\Sigma'^{-1}\mathbf{x}'/2) = \int d^n x \prod_{i=1}^{n} \exp(-x'^2_i/2\lambda_i)$$

$$\int d^n x' \prod_{i=1}^{n} \exp(-x'^2_i/2\lambda_i) \quad // d^n x = d^n x' \text{ since Jacobian=1 for rotation } \mathbf{x}' = R^{-1}\mathbf{x}$$

$$= \Pi_{i=1}^{n} \{ \int_{-\infty}^{\infty} dx'_i \exp(- x'^2_i/2\lambda_i) \}$$

$$= \Pi_{i=1}^{n} \{ [2\pi\lambda_i]^{1/2} \}$$

$$= (2\pi)^{n/2} (\Pi_{i=1}^{n} \lambda_i^{1/2}) .$$

The determinant of a diagonal matrix is the product of the diagonal elements, so

$$\det(\Sigma') = \Pi_{i=1}^{n} \lambda_i = (\Pi_{i=1}^{n} \lambda_i^{1/2})(\Pi_{i=1}^{n} \lambda_i^{1/2}) = (\Pi_{i=1}^{n} \lambda_i^{1/2})^2$$

$$\Rightarrow \quad (\Pi_{i=1}^{n} \lambda_i^{1/2}) = [\det(\Sigma')]^{1/2} .$$

Thus we have shown that

$$1/N = (2\pi)^{n/2} [\det(\Sigma')]^{1/2}$$

or

$$N = (2\pi)^{-n/2}[\det(\Sigma')]^{-1/2}$$

From above we had

$$\Sigma' = R^{-1}\Sigma R$$

$$\Rightarrow \quad \det(\Sigma') = \det(R^{-1}\Sigma R) = \det(R^{-1})\det(\Sigma)\det(R) = \det(\Sigma)$$

which says that $\det(\Sigma)$ is a "scalar" since the above line shows it is invariant under rotation R^{-1}. Our final result is then

$$N = (2\pi)^{-n/2}[\det(\Sigma)]^{-1/2} = (2\pi)^{-n/2} |\Sigma|^{-1/2}$$

where $|M|$ and $\det(M)$ are just two different ways to express the determinant of a matrix M.

Thus, the normalized multivariate Gaussian distribution is

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \exp(- [\mathbf{x}-\boldsymbol{\mu}]^T\Sigma^{-1}[\mathbf{x}-\boldsymbol{\mu}]/2)$$

as appears in the Ng lecture

2. Given a new example x, compute

$$p(x) = \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$$

3. A Verification Exercise

In order to verify the validity of p(**x**) as stated above, we should be able to compute the covariance matrix Σ from p(**x**) in this way

$$\Sigma_{ij} = E[(X_i-\mu_i)(X_j-\mu_j)] = \int d^n x\, (x_i-\mu_i)\,(x_j-\mu_j)\, p(\mathbf{x})$$

$$= \frac{1}{(2\pi)^{n/2}\,|\,\Sigma\,|^{1/2}} \int d^n x\, (x_i-\mu_i)\,(x_j-\mu_j) \exp(-[\mathbf{x}-\boldsymbol{\mu}]^T\Sigma^{-1}[\mathbf{x}-\boldsymbol{\mu}]/2)\,,$$

where the integral is over all of $\mathcal{R}^n$. Somehow that Σ^{-1} in the exponent is going to end up creating a Σ_{ij} on the left hand side. There are doubtless various clever ways to compute the integral shown above, but we shall proceed using mindless brute force.

The first step is to replace $\mathbf{x} - \boldsymbol{\mu} \rightarrow \mathbf{x}$ which is allowed since the integral is over all of $\mathcal{R}^n$. So the right hand side of our starting equation becomes (again, $d^n x \rightarrow d^n x$ since $d^n\mu=0$)

$$= \frac{1}{(2\pi)^{n/2}\,|\,\Sigma\,|^{1/2}} \int d^n x\, (x_i)\,(x_j) \exp(-\mathbf{x}^T\Sigma^{-1}\mathbf{x}/2)\,.$$

Next, change variables to x'_i as above so that $\mathbf{x} = R\mathbf{x}'$ where R is the special rotation which diagonalizes Σ into Σ'. Repeated indices are always implicitly summed in all that follows (but sometimes we show the sum anyway).

$$x_i = R_{ia}x'_a \qquad\qquad \mathbf{x}^T\Sigma^{-1}\mathbf{x} = \mathbf{x}'^T\Sigma'^{-1}\mathbf{x}' = \Sigma_{i=1}^{n}\lambda_i^{-1}x'^2_i$$

$$\int d^n x = \int d^n x' \qquad \text{since the "Jacobian" for a rotation in } \mathcal{R}^n \text{ is 1}$$

then the above becomes

$$= \frac{1}{(2\pi)^{n/2}\,|\,\Sigma\,|^{1/2}} \int d^n x'\; R_{ia}x'_a\, R_{jb}x'_b \exp(-\Sigma_{i=1}^{n}\lambda_i^{-1}x'^2_i/2)\,.$$

Extract the rotation factors and reduce clutter by replacing $\mathbf{x}' \rightarrow \mathbf{x}$ as integration variables

$$= \frac{1}{(2\pi)^{n/2}\,|\,\Sigma\,|^{1/2}}\; R_{ia}R_{jb} \int d^n x\; x_a\, x_b \exp(-\Sigma_{i=1}^{n}\lambda_i^{-1}x_i^2/2)\,.$$

Then expand the exponential into a product of same

$$= \frac{1}{(2\pi)^{n/2}\,|\,\Sigma\,|^{1/2}}\; R_{ia}R_{jb} \int d^n x\; x_a\, x_b \quad \Pi_{i=1}^{n} \exp(-\lambda_i^{-1}x_i^2/2)$$

If $a \neq b$, things can be rearranged this way

$$= \frac{1}{(2\pi)^{n/2}\ |\Sigma|^{1/2}}\ R_{ia}R_{jb}\ \ *$$

$$\{ \int dx_a\ x_a\ \exp(-\lambda_a^{-1}x_a^2/2) \}\{ \int dx_b\ x_b\ \exp(-\lambda_b^{-1}x_b^2/2) \}$$

$$* \Pi_{i\neq a,b} \left(\int dx_i\ \exp(-\lambda_i^{-1}x_i^2/2) \right) .$$

But this result is 0 because $\int dx_a\ x_a\ \exp(-\lambda_a^{-1}x_a^2/2) = 0$ (and same for b) because the integrand is odd, but the range of integration $(-\infty,\infty)$ is even. Thus, to get some non-zero result we must have a = b. So we continue along keeping only a = b

$$= \frac{1}{(2\pi)^{n/2}\ |\Sigma|^{1/2}}\ R_{ia}R_{ja}\ \ *$$

$$\{ \int dx_a\ x_a^2\ \exp(-\lambda_a^{-1}x_a^2/2) \}$$

$$* \Pi_{i\neq a} \left(\int dx_i\ \exp(-\lambda_i^{-1}x_i^2/2) \right) .$$

Look up the integrals to get

$$= \frac{1}{(2\pi)^{n/2}\ |\Sigma|^{1/2}}\ R_{ia}R_{ja}\ \ * \{ \sqrt{2\pi}\ \lambda_a^{3/2}\ \} * \Pi_{i\neq a}\ (\sqrt{2\pi}\,\lambda_i^{1/2})$$

$$= \frac{1}{(2\pi)^{n/2}\ |\Sigma|^{1/2}}\ R_{ia}R_{ja}\ \ (2\pi)^{n/2}\ \lambda_a\ \Pi_{i=1}^{n}\ (\lambda_i^{1/2})$$

$$= \frac{1}{(2\pi)^{n/2}\ \det(\Sigma')^{1/2}}\ R_{ia}R_{ja}\ \ (2\pi)^{n/2}\ \lambda_a\ \det(\Sigma')^{1/2} \qquad // \det(\Sigma) = \det(\Sigma')$$

$$= R_{ia}R_{ja}\,\lambda_a\ = R_{ia}[\delta_{a,b}\,\lambda_a\,]R_{jb} = R_{ia}\Sigma'_{ab}\,R^{T}_{bj} = (R\Sigma'R^{-1})_{ij} \qquad // \Sigma' = R^{-1}\Sigma R$$

$$= \Sigma_{ij} \qquad // \text{Eureka!}$$

Thus we have verified that

$$\int d^n x\ (x_i-\mu_i)\ (x_j-\mu_j)\ p(\mathbf{x}) = \frac{1}{(2\pi)^{n/2}\ |\Sigma|^{1/2}}\ \int d^n x\ (x_i-\mu_i)\ (x_j-\mu_j) \exp(-[\mathbf{x}-\boldsymbol{\mu}]^T\Sigma^{-1}[\mathbf{x}-\boldsymbol{\mu}]/2)$$

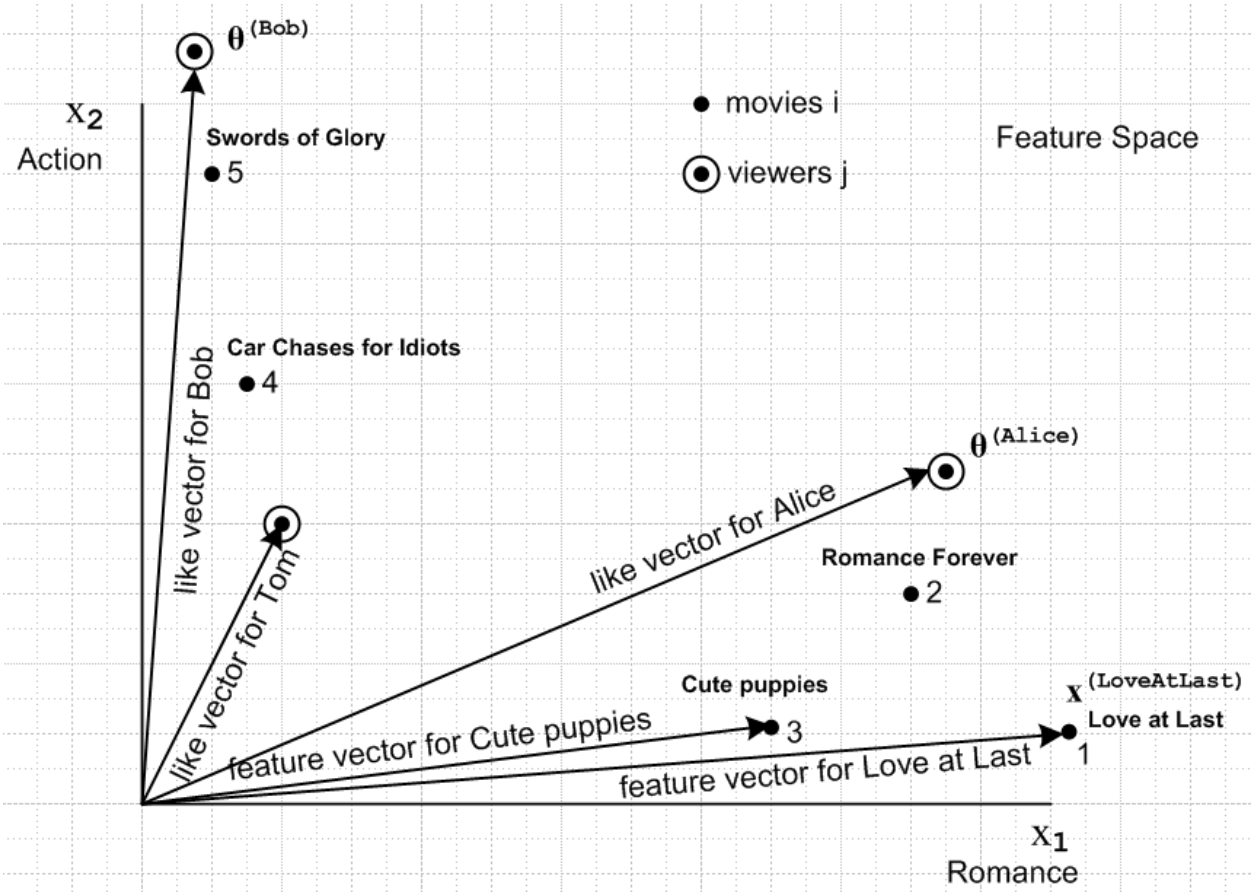
$$= \Sigma_{ij} .$$

G. Notes on Recommender Systems

1. Content Recommender System.

Feature space contains a feature vector $\mathbf{x}^{(i)}$ for each movie i. In Prof. Ng's example, each movie has 2 features, Romance and Action. Perhaps some paid reviewer watches the 5 movies in the example and comes up with $\mathbf{x}^{(i)}$ for each movie.

Meanwhile, each viewer is presumed to have some "like vector" in this same space. The j^{th} viewer has some vector $\boldsymbol{\theta}^{(j)}$. Let's just pretend women like romantic movies and men like action movies (not as in the lecture example). Here then might be the lay of the land:

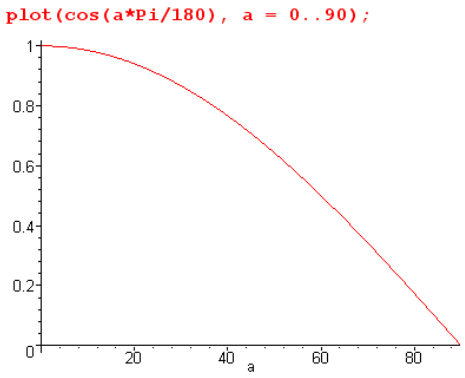


So each movie i has a vector $\mathbf{x}^{(i)}$, and each viewer j has a vector $\boldsymbol{\theta}^{(j)}$. Presumably a viewer will like a movie if his or her "like vector" $\boldsymbol{\theta}$ aligns with a movie's "feature vector" $\mathbf{x}$. The argument is that then the feature proportions of the movie match the feature proportions the user likes to see. So if Bob has not seen "Love at Last", we might predict that he won't like it much since the two vectors are so far apart. The amount of compatibility between a user and a movie in this model is given by the dot product of the two vectors: (a dot product of two vectors is maximal when the two vectors are aligned),

$$\text{how much Bob will like Love at Last} \;=\; \boldsymbol{\theta}^{(Bob)} \bullet \mathbf{x}^{(LoveAtLast)} \qquad // = \boldsymbol{\theta}^{(Bob)T}\, \mathbf{x}^{(LoveAtLast)}$$

$$= \| \boldsymbol{\theta}^{(Bob)} \| \| \mathbf{x}^{(LoveAtLast)} \| \cos(\text{angle between } \boldsymbol{\theta}^{(Bob)} \text{ and } \mathbf{x}^{(LoveAtLast)}) .$$

So how much Bob will like movie Love depends on the angle between the two vectors, *and* on the magnitude of each vector. If Love at Last is really, really romantic and has no action, it is way off to the right on the plot. If Bob really, really likes Action movies and hates Romantic movies, his arrow is long and near the vertical axis as shown. Here is a plot of the cos of the angle between two vectors (0 to 90 degrees)



so the limiting case is almost realized between Bob and Love at Last where the angle might be 80 degrees and cos(80deg) ≈ 0.2 from the plot.

So this is then the model prediction of how much Bob will like Love at Last :

$$y = \boldsymbol{\theta}^{(Bob)} \bullet \mathbf{x}^{(LoveAtLast)}$$

More generally, the model says $Y^{(i,j)}$ is the predicted rating viewer j will give movie i where

$$Y^{(i,j)} = \boldsymbol{\theta}^{(j)} \bullet \mathbf{x}^{(i)} .$$

If a viewer has already rated a movie, we can call that

$$y^{(i,j)} = \boldsymbol{\theta}^{(j)} \bullet \mathbf{x}^{(i)} .$$

Perhaps because of the possible confusion of the geometry of the dot product, Prof. Ng did not present the geometric picture shown above which I must say helps me a lot to see what is going on.

In the first part of the lecture, it is assumed that each movie has a pre-determined feature vector $\mathbf{x}^{(i)}$, and then the problem is to find the "like vectors" $\boldsymbol{\theta}^{(j)}$ for the viewers. The plan is to have viewers rate a few movies and then use their ratings to create their like vectors. This is done using the following linear regression method based on the usual least squares error,

$$J = \text{cost function for viewer j} = \Sigma_{i:\text{rated by j}} \, (\boldsymbol{\theta}^{(j)} \bullet \mathbf{x}^{(i)} - y^{(i,j)})^2$$

where the sum is over only those movies which viewer j has rated. Again, $y^{(i,j)}$ is the rating, $\mathbf{x}^{(i)}$ is the feature vector of the movie, and $\boldsymbol{\theta}^{(j)}$ is the like vector for viewer j which we seek to determine. To

minimize the cost, one could compute the derivatives of the above cost function and set them to zero to get a normal equation (see Section A), or one could use those derivatives in gradient descent. Since there are n features, there are going to be n derivatives for this one viewer j: $\partial J/\partial \theta^{(j)}_k$ for k = 1,2...n.

Instead of doing this for each viewer separately, Prof. Ng suggests doing it for all viewers at the same time. Then we add the squared error for all viewers to get this total cost function,

$$J_{total} = \Sigma_j \; \Sigma_{i:rated} \; (\theta^{(j)} \bullet x^{(i)} - y^{(i,j)})^2$$

and then one does this larger regression problem to find a solution vector $\theta^{(j)}$ for each viewer. The company doing this work (Netflix, say) really wants to minimize the total error over all movies and all users. The notation $\Sigma_{i:rated\ by\ j}$ is written $\Sigma_{i:r(i,j)=1}$ in the slides where r(i,j) = 1 if viewer j has rated movie i (and then the rating is $y^{(i,j)}$), else r(i,j) = 0.

So here is the final algorithm for gradient descent,

Optimization algorithm:

$$\min_{\theta^{(1)},\ldots,\theta^{(n_u)}} \frac{1}{2}\sum_{j=1}^{n_u}\sum_{i:r(i,j)=1}\left((\theta^{(j)})^T x^{(i)} - y^{(i,j)}\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{n_u}\sum_{k=1}^{n}(\theta_k^{(j)})^2$$

$J(\theta^{(1)},\ldots,\theta^{(n_u)})$

Gradient descent update:

$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \sum_{i:r(i,j)=1}((\theta^{(j)})^T x^{(i)} - y^{(i,j)})x_k^{(i)} \quad (\text{for } k = 0)$$

$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha\left(\sum_{i:r(i,j)=1}((\theta^{(j)})^T x^{(i)} - y^{(i,j)})x_k^{(i)} + \lambda\theta_k^{(j)}\right) \quad (\text{for } k \neq 0)$$

$\frac{1}{m^{(j)}}$ $\quad \frac{\partial}{\partial \theta_k^{(j)}} J(\theta^{(1)},\ldots,\theta^{(n_u)})$

where the usual regularization term has been added. Here those terms prevent the "like vectors" $\theta^{(j}$ from being too long. A very long $\theta^{(j)}$ would result in $y^{(i,j)} = \theta^{(j)} \bullet x^{(i)}$ being very large when these two vectors line up, and perhaps that results in a high-variance histogram as one holds j constant while varying i. In regular or logistic regression, high $\theta^{(j)}$ resulted in an oscillatory fit or decision boundary.

We noted above the *predicted* rating a viewer j would give movie i,

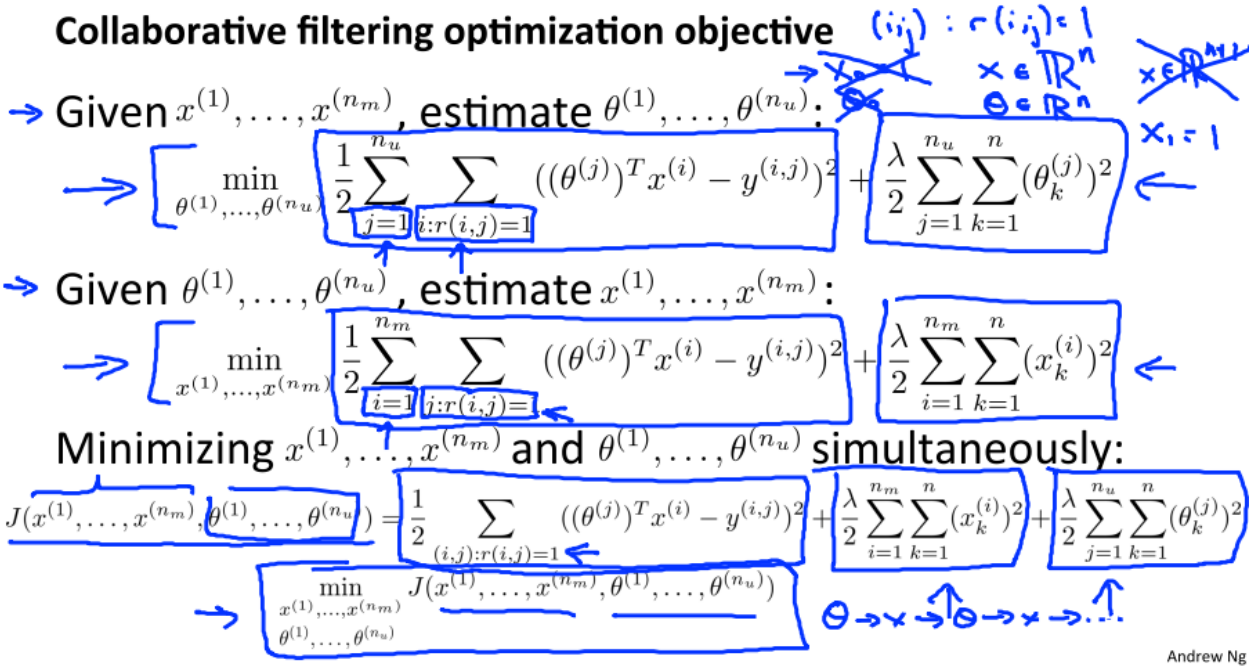
$$Y^{(i,j)} = \theta^{(j)} \bullet x^{(i)} .$$

Once the above regression is completed, one has $\mathbf{x}^{(i)}$ for every movie, and one has $\boldsymbol{\theta}^{(j)}$ for every viewer, and for viewer j one can compute the above $Y^{(i,j)}$ for all movies in stock and perhaps recommend to the viewer those movies which have a very high value of $Y^{(i,j)}$.

The above plan is called a **content-based recommender** system.

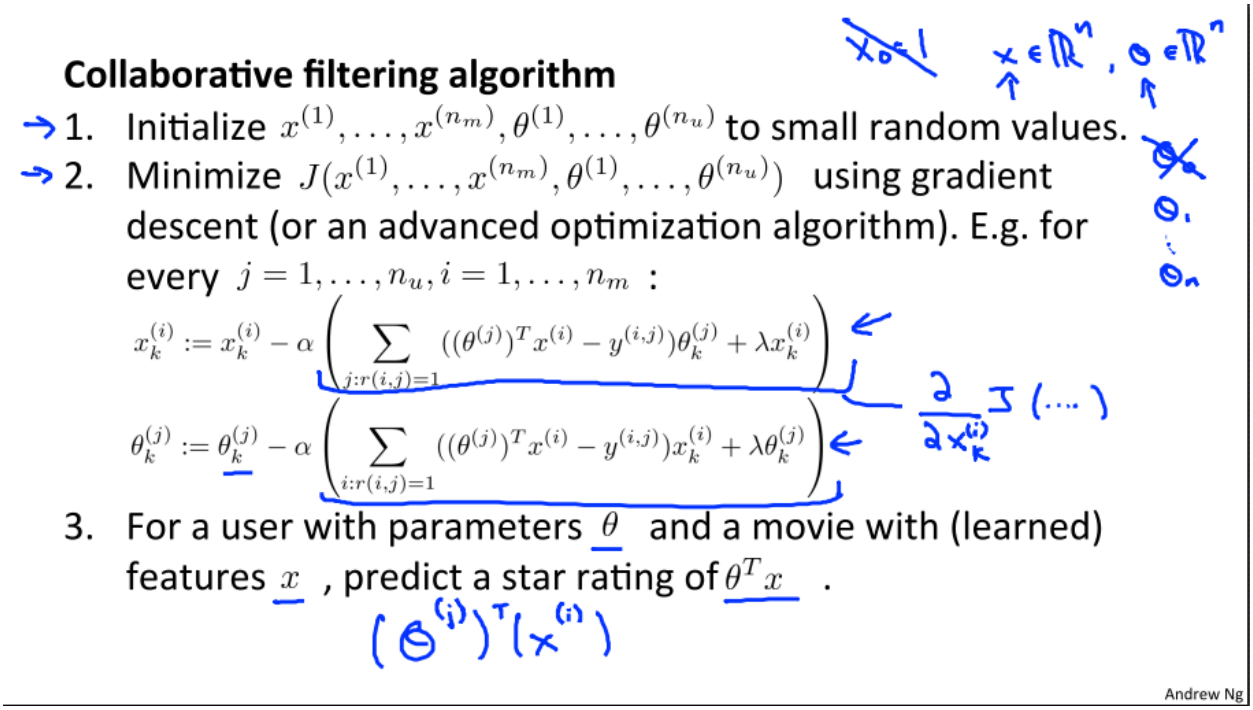
2. Collaborative Filtering Algorithm

Prof. Ng explains another system where, instead of hiring someone to produce values of the feature vector $\mathbf{x}^{(i)}$ for each movie i, one lets the algorithm determine both the feature components AND the like vectors. At first this is presented as an alternating iteration of phases where in one phase one assumes the $\mathbf{x}^{(i)}$ and then does regression to find the $\boldsymbol{\theta}^{(j)}$ and then in a second phase one assumes the $\boldsymbol{\theta}^{(j)}$ and then does regression to find the $\mathbf{x}^{(i)}$. He shows how you might as well do this as one huge regression problem and solve for all the $\mathbf{x}^{(i)}$ and $\boldsymbol{\theta}^{(j)}$ at the same time. Here is the slide which mushes the phases into one algorithm:



You do have to at least select the number of components n that a feature vector will have. This number is distinct from n_u which is the number of viewers (users) and n_m which is the number of movies. Presumably one could optimize n using the cross-verification type methods described in earlier lectures.

The final algorithm is then this:



One can write the predictions

$$Y^{(i,j)} = \theta^{(j)} \bullet x^{(i)} = \theta^{(j)T} x^{(i)}$$

as an n_m x n_u matrix Y where

$$Y_{ij} = Y^{(i,j)} = \theta^{(j)} \bullet x^{(i)} = \Sigma_{k=1}^{n} \theta^{(j)}_k x^{(i)}_k \ .$$

And one could then define matrix Θ and X as

$X_{ik} = x^{(i)}_k$ X has n_m rows and n columns
$\Theta_{jk} = \theta^{(j)}_k$ Θ has n_u rows and n columns

and then

$$Y_{ij} = \Sigma_{k=1}^{n} \Theta_{jk} X_{ik} = \Sigma_{k=1}^{n} X_{ik} \Theta^{T}_{kj}$$

or

$$Y = X\Theta^T$$

or

Y = X Θ^T .
n_m x n_u = n_m x n n x n_u

The rank of a non-square matrix cannot exceed the smaller dimension, so assuming the number of features n is smaller than the number of movies and number of users, both X and Θ^T have ranks which are $\leq$ n, so they both have "low rank". A matrix theorem states that

$\text{rank}(AB) \leq \text{rank}(A)$
$\text{rank}(AB) \leq \text{rank}(B)$

so matrix Y, although it might be quite large in both dimensions, has rank $\leq$ n .

Prof. Ng notes that if a viewer likes a movie with some feature vector $\mathbf{x}^{(i)}$, that viewer might also like a movie with a feature vector close to that vector, so such movies could be recommended. This is certainly compatible with the recommendation plan noted above.

Finally, if a viewer j has never rated a movie, the regularization term drives that users $\boldsymbol{\theta}^{(j)}$ to 0 so for all movies the predicted rating would be $Y^{(i,j)} = \boldsymbol{\theta}^{(j)} \bullet \mathbf{x}^{(i)} = 0$. This problem is repaired by adding to the dot product the mean of the ratings others have given that movie.

H. Notes on mini-batch, stochastic, online learning, and map-reduce.

The general topic here is dealing with very large training data sets, such as m = 100,000,000. One approach is that maybe the learning curves of an earlier lecture will show that only 1000 of these samples are really needed to create a useful solution weight vector $\boldsymbol{\theta}$ for a fitting or classification problem. If the large m is really necessary, the stochastic and mini-batch methods can reduce computation time, as can the map-reduce method of parallel processing.

1. Comparison of Algorithms.

Below is a comparison of the batch, mini-batch and stochastic algorithms. Bolded vectors have n components where n is the number of features. In each case, the "step" quantity used in the gradient descent algorithm -- here called **Q** -- is different:

$\boldsymbol{\theta} = \boldsymbol{\theta}^{(init)}$; // **batch** (b = m)
Repeat while $\boldsymbol{\theta}$ is still moving significantly {
  $\mathbf{Q} := (1/m)\Sigma_{i=1}^{m} (h_{\theta}(x^{(i)}) - y^{(i)})\, \mathbf{x}^{(i)}$; // this is $\partial J/\partial\theta_j$
  $\boldsymbol{\theta} := \boldsymbol{\theta} - \alpha\, \mathbf{Q}$;
}

$\boldsymbol{\theta} = \boldsymbol{\theta}^{(init)}$; // **mini-batch** (b = b)
Repeat while $\boldsymbol{\theta}$ is still moving significantly {
  for β = 1 to N do { // for each batch do
    $\mathbf{Q} := (1/b)\Sigma_{i=(\beta-1)b+1}^{\beta b} (h_{\theta}(x^{(i)}) - y^{(i)})\, \mathbf{x}^{(i)}$; // this is $\partial J^{(\beta)}/\partial\theta_j$
    $\boldsymbol{\theta} := \boldsymbol{\theta} - \alpha\, \mathbf{Q}$;
  }
} // for β = 1, get i = 1,2..b; for β =2, get i = b+1, b+2....2b, etc.

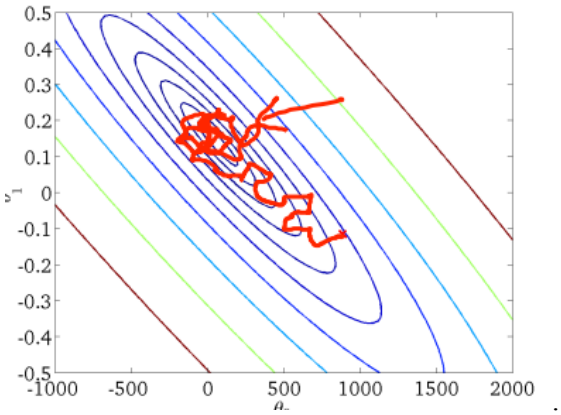
$\boldsymbol{\theta} = \boldsymbol{\theta}^{(init)}$; // **stochastic** (b = 1)
Repeat while $\boldsymbol{\theta}$ is still moving significantly {
  for i = 1 to m do { // for each sample do
    $\mathbf{Q} := (h_{\theta}(x^{(i)}) - y^{(i)})\, \mathbf{x}^{(i)}$; // this is $\partial J^{(i)}/\partial\theta_j$
    $\boldsymbol{\theta} := \boldsymbol{\theta} - \alpha\, \mathbf{Q}$;
  }
}

Only the full batch method is "correct" because only it properly minimizes the total cost J with respect to the weight parameters θ_j. The other two methods are hopeful approximations. The stochastic method is just the mini-batch method with b = 1.

The justification for the mini-batch method is this: One starts with the first mini-batch of b samples, and one in effect computes the average squared error for just that batch $J^{(\beta)}$ and computes the derivative which is $\partial J^{(\beta)}/\partial\theta_j$. That derivative is used to update $\boldsymbol{\theta}$. Then that updated $\boldsymbol{\theta}$ is used as the initial $\boldsymbol{\theta}$ for the next batch. So to the extent the first batch gave a good value of $\boldsymbol{\theta}$, the second batch has a leg up since it

starts with an initial value which is pretty good. This assumes that there is some similarity between the batches, and that might be a good reason to **pre-shuffle** (randomize) the data samples at the very start.

The stochastic case is just the mini-batch case with batch size set to b = 1. It is a bit hard to imagine how this case could work very well since one would not expect adjacent batches of 1 sample to be very similar. One would expect **θ** to bounce around rather violently in θ-space, but presumably it can eventually calm down and converge to some region near the correct solution,



The random appearance of the convergence path (sort of the reverse of the drunk walking away from a lamp post) is probably the basis of the name "stochastic gradient descent".

Prof. Ng suggested that during the running of the stochastic or mini-batch algorithms, one should record the individual sample (or mini-batch) costs J,

$J = (h_{\theta}(x^{(i)}) - y^{(i)})^2$ // stochastic

$J = \Sigma_{i=(\beta-1)b+1}^{\beta b} (h_{\theta}(x^{(i)}) - y^{(i)})^2$ // mini-batch

and then every so many samples (or mini-batches) plot the average of these samples, just to make sure the cost is in some sense going down. This is just the logical extension of the **convergence checking** plan outlined for the full batch process in an earlier lecture.

2. Comparison of Computation Costs.

Assume m = 300,000,000 or some large number. If each sample has 1 KB of associated data, then the dataset requires 300 GB which means it is going to be on a hard drive and not in memory (at least not on a routine computer).

In the full **batch** method, one has to serially access this entire data base S times, where S is the number of steps the gradient descent process requires (the number of Repeats). So for each of these passes through the data, there are S disk I/O times and then S*m*n computations $+= (h_{\theta}(x^{(i)}) - y^{(i)})\, x^{(i)}_j$, the total cost of which depends on the nature of $h_{\theta}(x)$ which is at least $\mathbf{\theta \bullet x} = \theta^T x = \Sigma_{k=1}^{n} \theta_k x_k$. The cost of doing the S updates $\mathbf{\theta} := \mathbf{\theta} - \alpha\, \mathbf{Q}$ is totally negligible.

In the **stochastic** method, only a single serial pass is required through the data (per Repeat) and there will be m descent steps per Repeat cycle. Prof. Ng comments that R=1 repeat passes might be enough if the **θ** values are stabilizing toward the end of the first pass, otherwise you continue for another pass which doubles computation time. Perhaps you break out in the middle of a pass if things become stable (within some small "ball" in θ space). Roughly there are R*m*n computations += $(h_{\boldsymbol{\theta}}(x^{(i)}) - y^{(i)})\, x^{(i)}_j$. If R = 1 works, then this method saves S-1 disk I/O times and reduces computation by 1/S. But to this we have to add back the cost of **θ** : = **θ** $- \alpha$ **Q** which is R*m*n multiply & add, but this is still small compared to the cost just noted.

In the **mini-batch** method, again only a single pass is required through the data (per Repeat) since a batch worth of data can be buffered in memory. The **Q** computation can then be done with a fast vectorized library call. There are N descent steps per Repeat cycle where N = m/b. Again, maybe only a single Repeat cycle will be required. Prof. Ng likes b = 10, and says b = 2-100 is the typical range. The cost here is again R*m*n computations += $(h_{\boldsymbol{\theta}}(x^{(i)}) - y^{(i)})\, x^{(i)}_j$, and the cost of **θ** : = **θ** $- \alpha$ **Q** is in this case R*N*n = R*(m/b)*n multiply & add, reduced 1/b from the stochastic case. The main saving here is the reduction of S-1 disk I/O passes (if R=1 works) , the reduction of computation by factor 1/S (again, if R=1 works) *and* the benefit of fast vectorized computation which helps for any R.

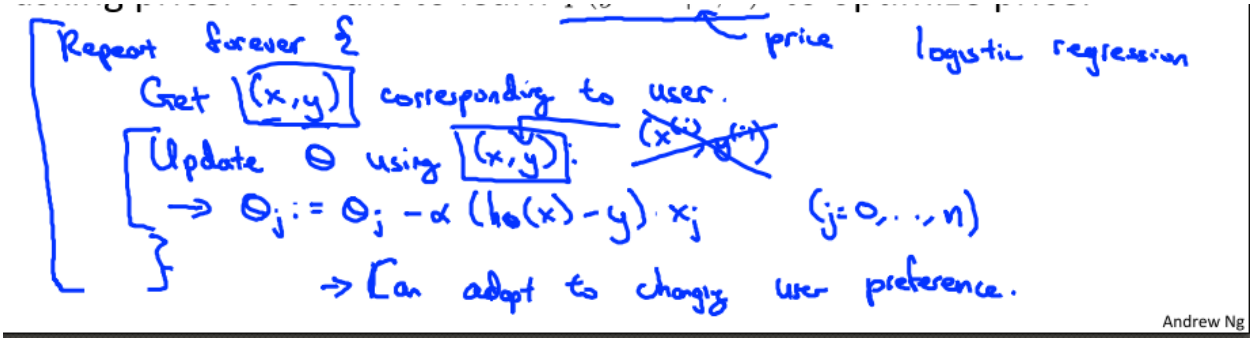
3. On Line Learning.

In the simplest form, this is just the stochastic case shown above, but modified as follows:

```
θ = θ(init);                                                    // stochastic ( b = 1)
Repeat forever {
    for i = 1 to ∞ do {                  // for each sample do
        wait for next user sample (x(i), y(i)) to come in;
        Q : = (hθ(x(i)) - y(i)) x(i) ;
        θ : = θ – α Q;
    }
}
```

or dispensing with the (i) label and the i loop that does nothing,

```
θ = θ(init);                                                    // stochastic ( b = 1)
Repeat forever {
        wait for next user sample (x, y) to come in;
        Q : = (hθ(x) - y) x;
        θ : = θ – α Q;
}
```



As Prof. Ng notes, this algorithm can account for changing user preferences, changing business conditions (a new aggressive competitor perhaps), and so on. In a typical application, the feature vector $\mathbf{x}^{(i)}$ might include a product description (shipping something from NY to LA, weight, time to ship) *and* the price of the product. Then $y^{(i)} = 1$ if user buys the product, else $y^{(i)} = 0$. One might use this online machine learning algorithm to assist in adjusting the product price to maximize profit over all products, presumably the goal of any commercial website.

4. Map-Reduce.

Prof. Ng's closing slides in this lecture show quite clearly how one can partition the training data base and have different computers, or different cores in the same CPU, each compute a share of the sums involved in machine learning. This is especially useful in the batch algorithm when m is very large. This technique has the unusual name "map reduce" and sometimes the multi-core version is implemented automatically within certain library routines where each sum contribution would be computed on a different thread on a different CPU core.