My Notes to Andrew Ng's Coursera "Machine Learning"

Eric Xia

Last Updated 8 September 2020

Contents

Week 1	Introduction Univariate Linear Regression Linear Algebra Review
Week 2	Multivariate Linear Regression Computing Parameters Analytically
Week 3	Logistic Regression Regularization
Week 4	Neural Networks: Representation
Week 5	Neural Networks: Learning
Week 6	Advice for Applying Machine Learning Machine Learning System Design
Week 7	Support Vector Machines (SVMs)
Week 8	Unsupervised Learning Dimensionality Reduction
Week 9	Anomaly Detection Recommender Systems
Week 10	Large Scale Machine Learning
Week 11	Application Example: Photo OCR

1 Introduction

Machine Learning (ML): the field of study that gives computers the ability to learn without being explicitly programmed
Learning Problems: a computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.

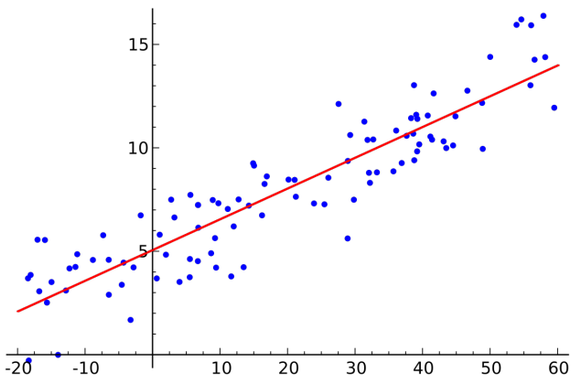
Example: Suppose your email program watches which emails you do or do not mark as spam, and based on that learns how to better filter spam. Identify the experience E, task T, and performance measure P in this scenario.
T: Classifying emails as spam or not spam.
E: Watching you label emails as spam or not spam.
P: The number (or fraction) of emails correctly classified as spam/not spam.

Supervised Learning: we give the computer a dataset and the right/wrong answers and then attempt to produce more correct answers.
Unsupervised Learning: we approach problems with little to no idea of what our results should look like and we derive structure from data where we don't necessarily know the effect of the variables; we let the computer learn itself, e.g. organizing computing clusters, social network analysis, market segmentation, astronomical data analysis.

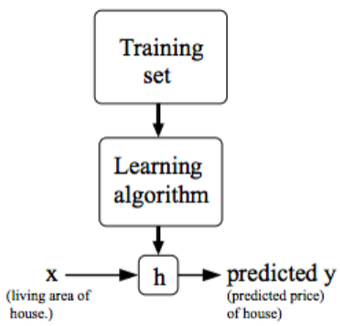
Clustering: finding natural groups in the feature space of input data
Regression: attempting to predict continuous values
Classification: attemping to predict discrete-valued output or map input variables into discrete categories, e.g. Benign vs. Malignant Tumor

2 Univariate Linear Regression

Training Set: the dataset used to train a model
$\boldsymbol{m}$: number of training examples
$\boldsymbol{x}$'s: input variables/features
$\boldsymbol{y}$'s: output variables/features
$(\boldsymbol{x}, \boldsymbol{y})$: one training example
$(\boldsymbol{x^{(i)}}, \boldsymbol{y^{(i)}})$: the ith training example
Hypothesis Function $(h : x \to y, h(x) = h_\theta(x) = \theta_0 + \theta_1 x)$: given a training set, we want to choose **parameters** (θ_0 **and** θ_1)such that the mean distance between $h(x)$ and the training examples are as close to the true mean distance:



Linear Regression



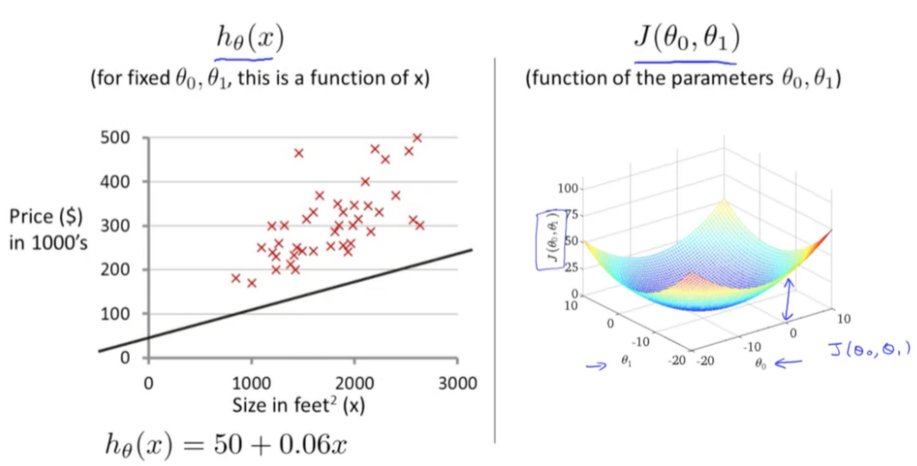
Hypothesis Function

We can measure the error of a hypothesis function by using a **cost function** J, given by

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2$$

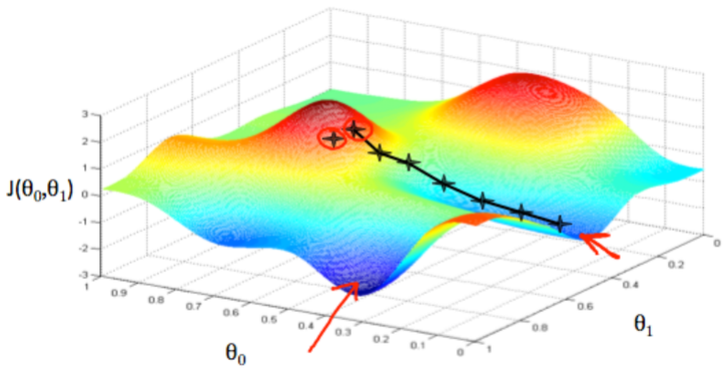
This function is really just the average of the differences between the produced outputs with the hypothesis function being tested, and the actual outputs. The cost function is also referred to as the squared error function. We half the mean in the cost function out of convenience for computing the gradient descent, since the derivative term of the square function will cancel out the half.

The contour plot display of the cost function for a particular hypothesis function:



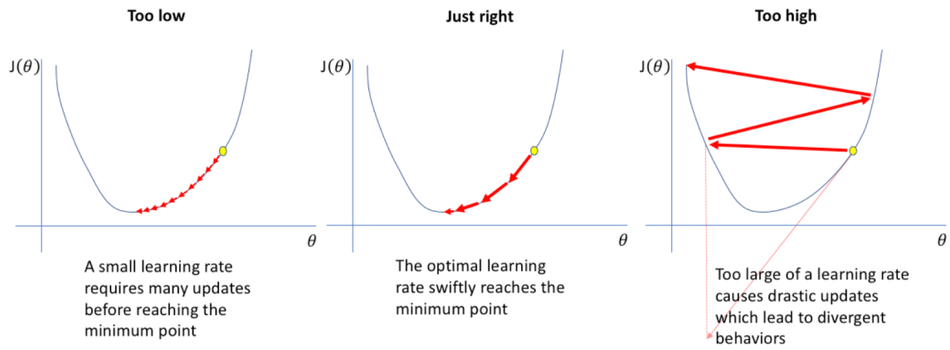
Gradient Descent: an optimization algorithm used to minimize some function, e.g. the cost function, by iteratively moving in the direction of the steepest descent as defined by the negative of the gradient.

Putting θ_0 on the x-axis and θ_1 on the y-axis, with the cost function on the z-axis, the points on the graph are the outputs of our cost function. We know we have succeeded in finding parameters for $J(\theta_0, \theta_1)$ that minimize the cost when our gradient descent leads us to the blue area of the graph (lowest z-value). The red arrows on the graph show the minimum points on the graph.



We implement this algorithm by taking the derivative of the cost function. This gives us the slope of the cost function at a point, and a direction to move towards. We then iteratively step down the cost function in the direction of the steepest descent. The size of each step is determined by the **learning rate**, the parameter α. Note that α is fixed and its value need not variate because the cost function is parabolic, hence as we approach the minimum, the derivative decreases in magnitude and flattens out.

We want to choose a value of α that is not too small or too large:

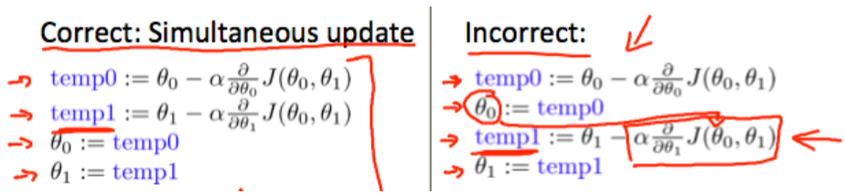


Then our implementation of gradient descent is given by repeating the following equation until convergence.

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

where $j = 0, 1$ represents the index and ":=" is the assignment operator.

Note that the gradient descent equations for θ_0 and θ_1 should be updated simultaneously, such that updating of θ_0 does not interfere with the accurate updating of θ_1:



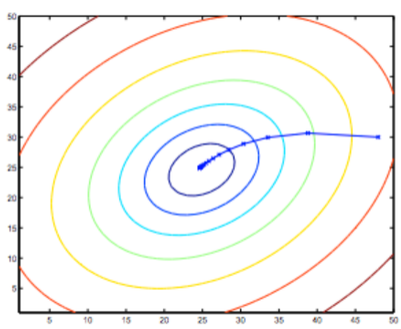
The current gradient descent algorithm we are using is called **batch gradient descent** because it accounts for every training example to determine the descent steps. For a single training example,

$$\begin{aligned}
\frac{\partial}{\partial \theta_j} J(\theta) &= \frac{\partial}{\partial \theta_j} \frac{1}{2} (h_\theta(x) - y)^2 \\
&= 2 \cdot \frac{1}{2} (h_\theta(x) - y) \cdot \frac{\partial}{\partial \theta_j} (h_\theta(x) - y) \\
&= (h_\theta(x) - y) \cdot \frac{\partial}{\partial \theta_j} \left(\sum_{i=0}^{n} \theta_i x_i - y \right) \\
&= (h_\theta(x) - y) x_j
\end{aligned}$$

Hence, we repeat the following equations until convergence.

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)$$
$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x_i) - y) x_i$$

While gradient descent can be susceptible to local minima, simple optimization problems for linear regression only have one global, and no other local or optima. Thus, gradient descent will always converge here to the global minimum. Assuming we have a well-chosen α, we see that J is indeed a convex quadratic function, and the below contour plot depicts the trajectory of an implementation of gradient descent to minimize a quadratic function. The points start from the warm outermost colors and move towards the cooler innermost colors, until they converge at the global minimum of the cost function.



3 Linear Algebra Review

Matrix: a 2D array
- The dimension of a matrix is given by "number of rows × number of columns"
- Dimension may be written $\mathbb{R}^{4\times2}$, representing a 4×2 matrix of the real-valued set $\mathbb{R}$
- In the declaration below, A_{iy} = "i, j entry" in the ith row, jth column

$$A = \begin{bmatrix} 1402 & 191 \\ 1371 & 821 \\ 949 & 1437 \\ 147 & 1448 \end{bmatrix}$$

Vector: A $n \times 1$ matrix
- Can be 1-indexed or 0-indexed, where 1-indexed is more conventional
- It is conventional to name matrices with capital letters and vectors with lowercase letters
- In the declaration below, $y_i = i$th element of the given 4D Vector, $\mathbb{R}^4$

$$y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}$$

Matrix Addition: a defined matrix sum must have two addends with the same dimensions.

$$\begin{bmatrix} a_1 & d_1 \\ b_1 & e_1 \\ c_1 & f_1 \end{bmatrix} + \begin{bmatrix} a_2 & d_2 \\ b_2 & e_2 \\ c_2 & f_2 \end{bmatrix} = \begin{bmatrix} a_1 + a_2 & = d_1 + d_2 \\ b_1 + b_2 & = e_1 + e_2 \\ c_1 + c_2 & = f_1 + f_2 \end{bmatrix}$$

Scalar Matrix Multiplication:

$$C \begin{bmatrix} a & d \\ b & e \\ c & f \end{bmatrix} = \begin{bmatrix} Ca & Cd \\ Cb & Ce \\ Cc & Cf \end{bmatrix}$$

Division example:

$$\begin{bmatrix} 4 & 0 \\ 6 & 3 \end{bmatrix} /4 = \frac{1}{4} \begin{bmatrix} 4 & 0 \\ 6 & 3 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 3/2 & 3/4 \end{bmatrix}$$

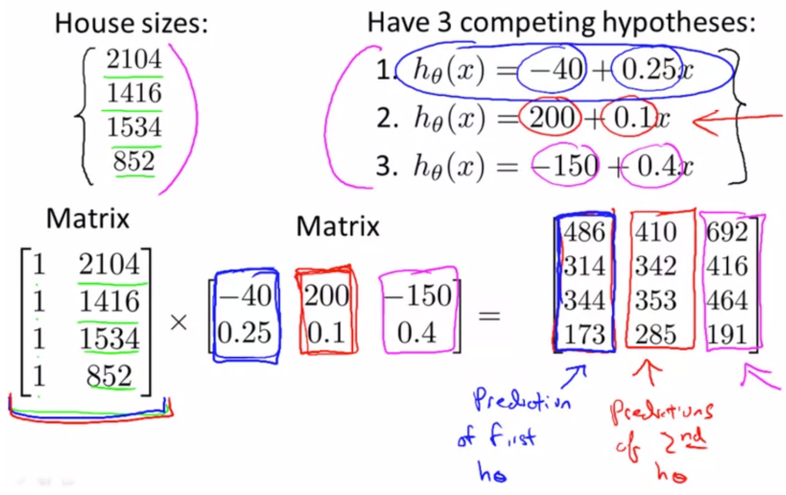
Matrix-Vector Multiplication: if A is a $m \times n$ matrix then the product Ax is defined by $n \times 1$ column vectors x. If $Ax = b$ then b is a $m \times 1$ column vector.

$$\text{Ax} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \end{bmatrix}$$

Example:

$$\begin{bmatrix} 1 & -1 & 2 \\ 0 & -3 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \cdot 1 & -1 \cdot 1 & 0 \cdot 2 \\ 2 \cdot 0 & -1 \cdot 3 & 0 \cdot 1 \end{bmatrix} = \begin{bmatrix} 1 \\ -3 \end{bmatrix}$$

Matrix-Matrix Multiplication to Combine Hypotheses: If we wanted to make a hypothesis from 3 hypotheses we could represent the hypotheses by matrices and the dataset by another matrix and multiply them to get the product matrix which would be the combined hypothesis:



Properties of Matrix-Matrix Multiplication:
• In general, $A \times B \neq B \times A$ (not commutative)
• $A \times (B \times C) = (A \times B) \times C$ (is associative)

Identity Matrix: a special square matrix that has 1's along the main diagonal (upper left to lower right) and 0's for all other elements
• Denoted I or I_n
• For any matrix $A, A_{m\times n} \cdot I_{n\times n} = I_{m\times m} \cdot A_{m\times n} = A_{m\times n}$ (the subscripts are usually omitted)
• $A = B$ when $B = I$ (matrices are commutative when at least one of them is the identity matrix)

Example of an identity matrix:

$$I_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Informally,

Matrix Inverse: If A is a $m \times m$ matrix, and if it has an inverse (only square matrices have inverses), then $A(A^{-1}) = A^{-1}A = I$

Matrix Transpose: If A is a $m \times n$ matrix, $B = A^T$ then B is a $n \times m$ matrix, and $B_{ij} = A_{ij}$
• Basically, the rows of A become the columns of B and the columns of A become the rows of B

$$A = \begin{bmatrix} 1 & 2 & 0 \\ 0 & 5 & 6 \\ 7 & 0 & 9 \end{bmatrix}, A^T = \begin{bmatrix} 1 & 0 & 7 \\ 2 & 5 & 0 \\ 0 & 6 & 9 \end{bmatrix}$$

4 Multivariate Linear Regression

Notation: $\boldsymbol{x_j^{(i)}}$ = value of features j in ith training example
$\boldsymbol{x^{(i)}}$ = input (features) of ith training example
$\boldsymbol{m}$ = the number of training examples
$\boldsymbol{n}$ = the number of features

We can extend the univarate hypothesis function, $h_\theta(x) = \theta_0 + \theta_1 x$ to multiple variables like so:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \cdots + \theta_n x_n$$

We can represent the **multivariate hypothesis function** by defining $x_0 = 1$ and writing the x's and θ's as parameter vectors:

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1}, \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^{n+1}$$

Then

$$\begin{aligned} h_\theta(x) &= \theta_0 x_0 + \theta_1 x_1 + \cdots + \theta_n x_n \\ &= \theta^T x \end{aligned}$$

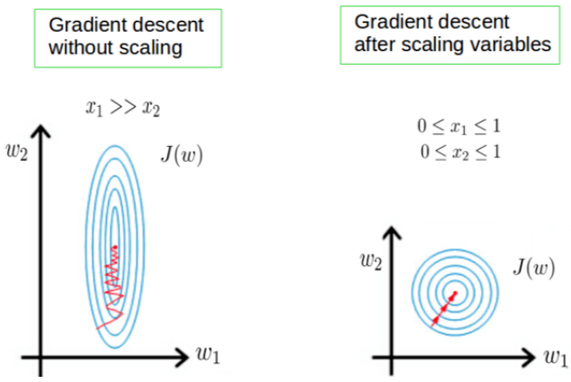
where $\theta^T = [\theta_0, \theta_1, \ldots, \theta_n]$ and $x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ n \end{bmatrix}$. Together, $\theta^T x$ is read "theta transpose x".

Multivariate Gradient Descent Equation:

$$\begin{aligned} \theta_j &:= \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \\ &:= \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \end{aligned}$$

where we simulatenously update θ_j for $j = 0, \ldots, n$. Note that the parameter of J, θ, represents a vector containing all the θ's from θ_0 to θ_n.

We can speed up gradient descent by having our input values in roughly the same range. This is because θ descends faster on small ranges and slowly on large ranges so it will oscillate inefficiently down to the minimum of the cost function when the variables are uneven. Ideally, we want $-1 \leq x_{(i)} \leq 1$ or $-0.5 \leq x_{(i)} \leq 0.5$. If the range is too large, say, $-3 \leq x \leq 3$ or too small, like $-\frac{1}{3} \leq x \leq \frac{1}{3}$, then the contours of our cost function will be really narrow and gradient descent will be more inefficient.



One way to prevent gradient descent inefficiency is by **feature scaling**, which involves dividing the input values by the range of the input variable, resulting in a new range of simply 1. The following equation demonstrates how we rescale a value of a features.

$$x_{\text{new}} = \frac{x_i - \min(x)}{\max(x) - \min(x)}$$

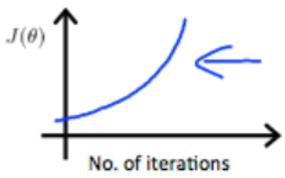
Mean Normalization is a second way we might rescale features, involving subtracting the average value for an input variable from the values for that input variable which result in a new average value for the input variable of simply 0. We adjust input values by the following formula.

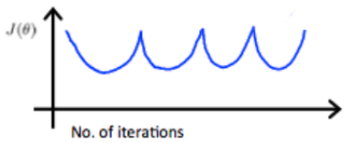
$$x_i := \frac{x_i - \mu_i}{s_i}$$

where μ_i is the mean of the feature values and s_i is the range of values or the standard deviation. For example, if x_i represents housing prices with a range of 100 to 2000 and a mean value of 1000, then $x_i := \frac{\text{price} - 1000}{1900}$.

We can debug gradient descent by making a plot with the number of iterations on the x-axis. Then plotting the cost function over the number of iterations of gradient descent, if $J(\theta)$ ever increases, then it is likely that we need to decrease the value of α. If α is too large, then the cost function may not decrease on every iteration and may diverge. However, if α is too small, the convergence may be too slow.

Common plots that show we need a smaller learning rate include the two figures below.





We can change the behavior of a hypothesis function by making it a different type of function, including quadratic, cubic, and radical if it fits the data better. If our hypothesis function is $h_\theta(x) = \theta_0 + \theta_1 x_1$ then we can create additional features based on x_1 to get the square root function $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 \sqrt{x_1}$ or the cubic function $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^3$.

5 Computing Parameters Analytically

Normal Equation: an analytical approach to Linear Regression with a Least Square Cost Function
• Allows us to directly find the parameters θ without using gradient descent.

A Derivation of the Normal Equation:
We have our hypothesis function $h(\theta) = \theta^T x$ and cost function $J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}[h_\theta(x^{(i)}) - y^{(i)}]^2$. Let us create a matrix X such that

$$X = \begin{bmatrix} x^1 \\ x^2 \\ \vdots \\ x^i \end{bmatrix}$$

where X is a $m \times 1$ matrix and each row is the ith training example. Now we can use X's definition to rewrite the cost function $J(\theta)$, ignoring the $\frac{1}{2m}$ because we're computing the derivative to zero so it doesn't matter. Then

$$\begin{aligned} J(\theta) &= ((X\theta)^T - y^T)(X\theta - y) \\ &= (X\theta)^T X\theta - (X\theta)^T y - y^T(X\theta) + y^T y \\ &= \theta^T X^T X\theta - 2(X\theta)^T y + y^T y \end{aligned}$$

Note that both $X\theta$ and y are vectors, hence, they have the associative property of multiplication and we were able to simplify down to the above equation. Since we want to minimize the cost function $J(\theta)$, let's set the partial derivative to 0.

$$\frac{\partial J(\theta)}{\partial \theta} = 0$$

We can ignore the last term of $J(\theta)$ because its behavior doesn't depend on the variable of interest. Let us differentiate the second term of $J(\theta)$, which we will refer to as

$$P(\theta) = 2(X\theta)^T y$$

Recall that θ is a vector of n components, y is a vector of m components, and X is a $m \times n$ matrix.
Then the matrix expansion of $P(\theta)$ is

$$P(\theta) = 2\left[\begin{pmatrix} x_{11} & x_{12} & \dots & x_{1n} \\ x_{21} & \dots & \dots & x_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ x_{m1} & \dots & \dots & x_{mn} \end{pmatrix}\begin{pmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{pmatrix}\right]^T \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix}$$

Then

$$\begin{aligned} P(x) &= 2\left[\begin{pmatrix} x_{11}\theta_1 + \dots + x_{1n}\theta_n \\ x_{21}\theta_1 + \dots + x_{2n}\theta_n \\ \vdots \\ x_{m1}\theta_1 + \dots + x_{mn}\theta_n \end{pmatrix}\right]^T \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix} \\ &= 2(x_{11}\theta_1 + \dots + x_{1n}\theta_n)y_1 + 2(x_{21}\theta_1 + \dots + x_{2n}\theta_n)y_2 + \dots + 2(x_{m1}\theta_1 + \dots + x_{mn}\theta_n)y_m \\ &= 2\sum_{i=1}^{m} y_i(x_{i1}\theta_1 + \dots + x_{in}\theta_n) \\ &= 2\sum_{i=1}^{m} y_i \sum_{j=1}^{n} x_{ij}\theta_j \end{aligned}$$

and

$$\begin{aligned} \frac{\partial P}{\partial \theta_1} &= 2(x_{11}y_1 + \dots + x_{m1}y_m) \\ \frac{\partial P}{\partial \theta_2} &= 2(x_{12}y_1 + \dots + x_{m2}y_m) \\ \frac{\partial P}{\partial \theta_n} &= 2(x_{1n}y_1 + \dots + x_{mn}y_m) \end{aligned}$$

Since $\frac{\partial P}{\partial \theta}$ can be thought of as a vector of n components, we can rewrite these equations using a matrix-by-vector multiplication, effectively getting

$$\frac{\partial P}{\partial \theta} = 2X^T y$$

We also now know that

$$\frac{\partial}{\partial\theta}(X\theta)^T y = X^T y$$

Going back to $J(\theta) = \theta^T X^T X\theta - 2(X\theta)^T y + y^T y$, let us define the first term as

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

We expand and transpose $Q(\theta)$ such that

$$Q(\theta) = (\theta_1 \dots \theta_n) \begin{pmatrix} x_{11} & x_{21} & \dots & x_{m1} \\ x_{12} & \dots & \dots & x_{m2} \\ \vdots & \vdots & \vdots & \vdots \\ x_{1n} & \dots & \dots & x_{mn} \end{pmatrix} \begin{pmatrix} x_{11} & x_{12} & \dots & x_{1n} \\ x_{21} & \dots & \dots & x_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ x_{m1} & \dots & \dots & x_{mn} \end{pmatrix} \begin{pmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{pmatrix}$$

Multiplying the two largest matrices together, we have a "X-squared matrix" which is $n \times n$. The rows r and columns c satisfy the declaration

$$X^2_{rc} = \sum_{i=1}^{m} x_{ir} x_{ic}$$

We multiply X^2_{rc} by the θ vector to get

$$\begin{aligned} Q(\theta) &= (\theta_1 \dots \theta_n) \begin{pmatrix} x^2_{11}\theta_1 + \dots + x^2_{1n}\theta_n \\ x^2_{21}\theta_1 + \dots + x^2_{2n}\theta_n \\ x^2_{n1}\theta_1 + \dots + x^2_{nn}\theta_n \end{pmatrix} \\ &= \theta_1(X^2_{11}\theta_1 + \dots + X^2_{1n}\theta_n) + \theta_2(X^2_{21}\theta_1 + \dots + X^2_{2n}\theta_n) + \dots + \theta_n(X^2 n1\theta_1 + \dots + X^2_{nn}\theta_n) \end{aligned}$$

We have

$$\frac{\partial Q}{\partial\theta_1} = (2\theta_1 x^2_{11\theta 2} X^2_{12} + \dots + \theta_n X^2_{21} + \dots + \theta_n X^2 n1)$$

Since X^2 is symmetric, we know that $X^2_{12} = X^2_{21}$, etc. Hence,

$$\begin{aligned} \frac{\partial Q}{\partial\theta_1} &= 2\theta_1 X^2_{11} + 2\theta_2 X^2_{12} + \dots + 2\theta_n X^2_{1n} \\ \frac{\partial Q}{\partial\theta} &= 2X^2\theta \\ \frac{\partial Q}{\partial\theta} &= 2X^T X\theta \\ \frac{\partial J}{\partial\theta} &= \frac{\partial Q}{\partial\theta} - \frac{\partial P}{\partial\theta} \\ &= 2X^T X\theta - 2X^T y \end{aligned}$$

We want to set the partial derivative to 0 in order to find the values of θ which minimize the cost function $J(\theta)$:

$$\begin{aligned} 2X^T X\theta - 2X^T y &= 0 \\ X^T X\theta &= X^T y \end{aligned}$$

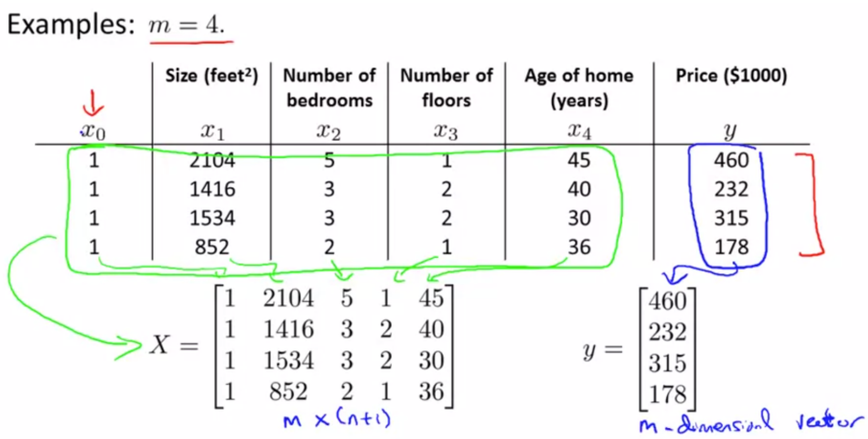
Assume the matrix $X^T X$ is invertible; we can multiply both sides of the above equation by $(X^T X)^{-1}$. Then we get the **normal equation,**

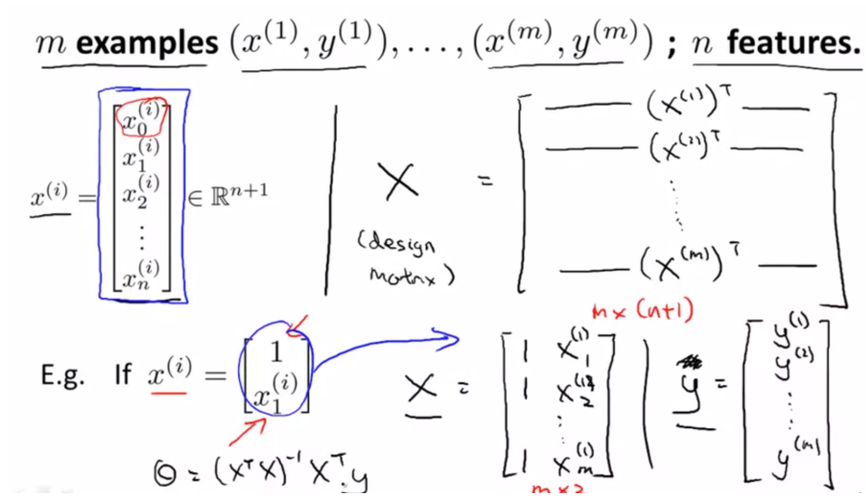
$$\theta = (X^T X)^{-1} X^T y$$

Below is an example of using the normal equation to quickly calculate the desired value of θ.

Examples: $m = 4$.

Size (feet²) x_1	Number of bedrooms x_2	Number of floors x_3	Age of home (years) x_4	Price ($1000) y
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178





Both gradient descent and using the normal equation have their advantages and disadvantages. As a general rule of thumb,

Gradient Descent	**Normal Equation**
Need to choose α	No need to choose α
Needs many iterations	No need to iterate
$O(kn^2)$	$O(n^3)$, need to calculate inverse of $X^T X$
Works well when n is large	Slow if n is very large

In practice, it might be a good time to start using an iterative process when $n > 10,000$.

In this class, we want to use the Octave "pinv" function instead of "inv", which will give a value of θ even if $X^T X$ is non-invertible. If $X^T X$ is non-invertible, common causes include redundant features, where two features are very closely related (linearly dependent), or too many features (e.g. $m \leq n$).