CS 6140: Machine Learning **Spring 2015**
College of Computer and Information Science
Northeastern University
Lecture 2 **January, 26**
Instructor: Bilal Ahmed **Scribe: Bilal Ahmed & Virgil Pavlu**

Linear Regression[1]

Consider the housing dataset where the objective is to predict the price of a house based on a number of features that include the average number of rooms, living area, pollution levels of the neighborhood, etc. In this example we are going to use a single feature: the average number of rooms to predict the value of the house. Our input is a single number $x \in \mathbb{R}$ and the output (label) is also a continuous number $y \in \mathbb{R}$, and our task is to find a function $f(x) : \mathbb{R} \to \mathbb{R}$ that takes as input the average number of rooms and outputs the value of the house (in tens of thousands of dollars). Here is a snippet of the data:

Average No. of Rooms	3	3	3	2	4	...
Price (10,000 $)	40.0	33 .0	36.9	23.2	54.0	...

and here is how the data looks like,

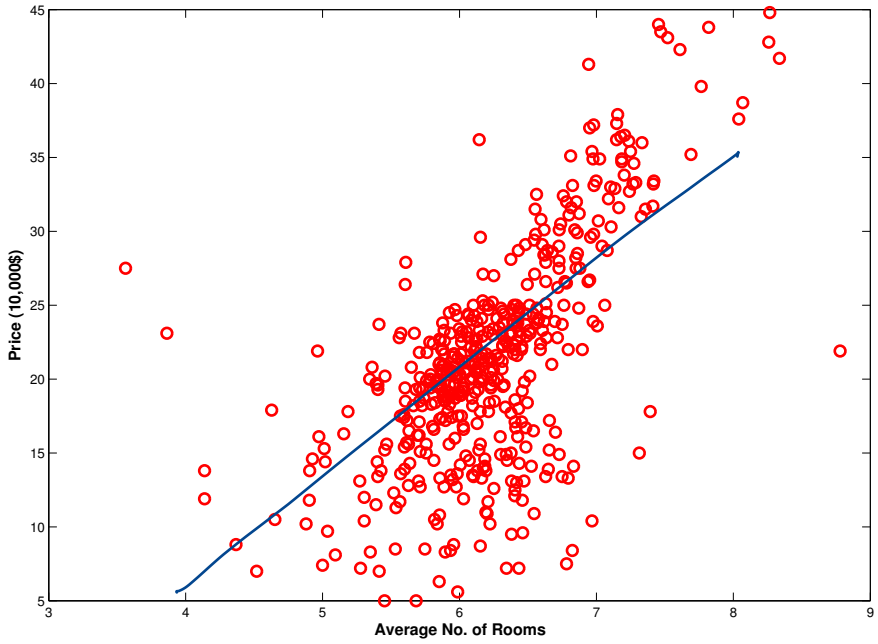


Figure 1: Plot of the average number of rooms (x-axis) and the price of the house (y-axis). Note that the input and the output are not normalized.

[1]Based on lecture notes by Andrew Ng. These lecture notes are intended for in-class use only.

Assume that the function that we are searching for is a straight line. In which case we can represent the function as:

$$y = f(x) = w_0 + w_1 x \quad (1)$$

where w_0 is the y-intercept of the line, and w_1 is the slope. Based on this formulation searching for the "best" line then translates into finding the optimal values of the parameters w_0 and w_1. It should be noted here that for a given training dataset having N observations, the x_i and y_i are fixed and we need to find the values of the parameters that satisfy the N equations:

$$\begin{aligned} y_1 &= w_0 + w_1 x_1 \\ y_2 &= w_0 + w_1 x_2 \\ &\dots \\ y_N &= w_0 + w_1 x_N \end{aligned}$$

If $N > 2$, then this system of equations is overdetermined and has no solution. Instead of looking for an exact solution that satisfies the N equations, we will search for an approximate solution, that satisfies these equations with some error. In order to find the approximate solution we need a way to decide the "goodness" of a given line (specific values for w_0 and w_1). For example, consider Figure-2, we have two candidate lines l_1 and l_2, which one should we choose? In other words how can we say that a given line is the optimal line with respect to the criterion we have defined for the approximate satisfiability of the set of equations given above?

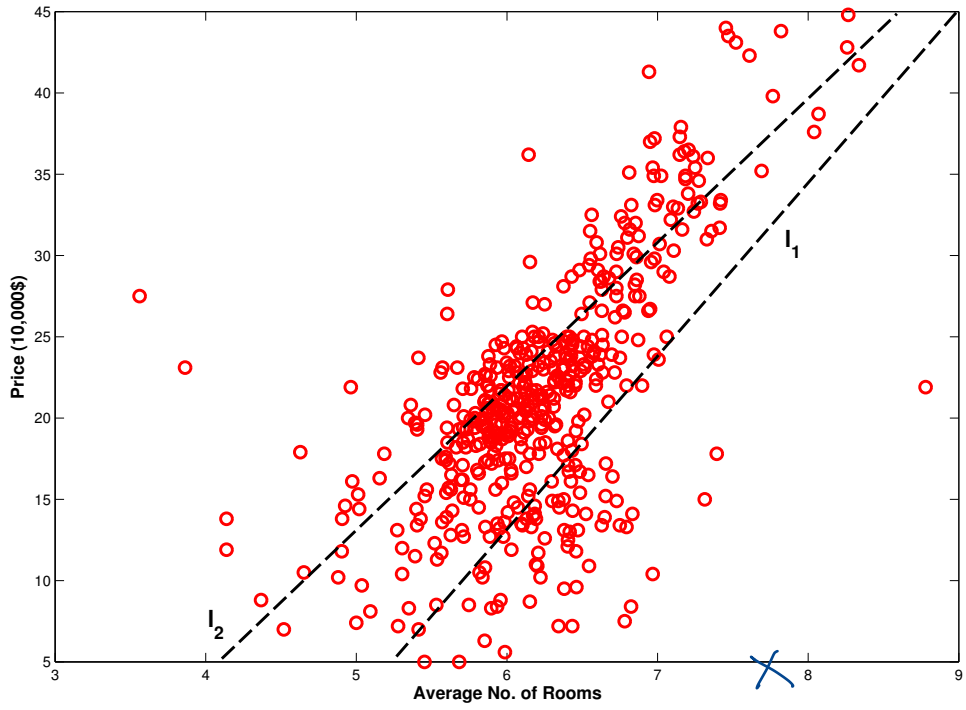


Figure 2: Two candidate lines, that can be used to predict the price of the house based on the average number of rooms. Which line is better?

1 Setup and Notation

So far in the example we have dealt with a single input feature and in most real-world cases we would have multiple features that define each instance. We are going to assume that our data is arranged in a matrix with each row representing an instance, and the columns representing individual features. We can visualize the data matrix as:

$$X = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1m} \\ x_{21} & x_{22} & \dots & x_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N1} & x_{N2} & \dots & x_{Nm} \end{bmatrix}$$

where, X is also known as the "design matrix". There are N labels corresponding to each instance $\mathbf{x_i} \in \mathbb{R}^m$, arranged as a vector $\mathbf{y} \in \mathbb{R}^N$ as:

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

the training data $\mathcal{D}$ can be described as consisting of $(\mathbf{x_t}, y_t)$; $\forall t \in \{1, 2, \dots, N\}$.

The regression function that we want to learn from the data can then be described analogously to Equation-1 as:

$$y = h_w(\mathbf{x}) = w_0 + w_1 x_1 + w_2 x_2 + \dots + + w_m x_m \tag{2}$$

where, $f(\mathbf{x}) : \mathbb{R}^m \to \mathbb{R}$ and $\mathbf{w} = [w_0, w_1, w_2, \dots, w_m]$ are the parameters of the regression function. The learning task is then to find the "optimal" weight vector $\mathbf{w} \in \mathbb{R}^{m+1}$, based on the given training data $\mathcal{D}$. When there is no risk of confusion, we will drop w from the f notation, and we will assume a dummy feature $x^0 = 1$ for all instances such that we can re-write the regression function as:

$$f(\mathbf{x}) = w_0 x_0 + w_1 x_1 + w_2 x_2 + \dots + w_m x_m = \sum_{i=1}^{m} w_i x_i = \mathbf{w}^T \mathbf{x} \tag{3}$$

Note: In the above equation x_i is the i^{th} feature, and to incorporate the augmented constant feature (which is always equal to one) we can augment the design matrix X with a column of 1s (as the first column) if $\mathbf{w} = [w_0, w_1, w_2, \dots, w_m]$.

2 Cost Function:

What do we mean by the best regression fit? For a given training dataset $\mathcal{D}$ and a parameter vector $\mathbf{w}$, we can measure the error (or cost) of the regression function as the sum of squared errors (SSE) for each $\mathbf{x_i} \in \mathcal{D}$:

$$E_w = \sum_{i=1}^{N} (f_w(\mathbf{x_i}) - y_i)^2 = \sum_{i=1}^{N} (\mathbf{w}^T \mathbf{x}_i - y_i)^2$$

the error function measures the squared difference between the predicted label and the actual label. The goal of the learning algorithm would then be to find the parameter vector $\mathbf{w}$ that minimizes the error on the training data. The optimal parameter vector $\mathbf{w}^*$ can be found as:

$$\mathbf{w}^* = \arg\min_w J(\mathbf{w}) = \arg\min_w \sum_{i=1}^{N} (\mathbf{w}^T \mathbf{x}_i - y_i)^2 \tag{4}$$

3 Minimizing a Function

Consider the function:

$$f(x) = 3x^2 + 4x + 1$$

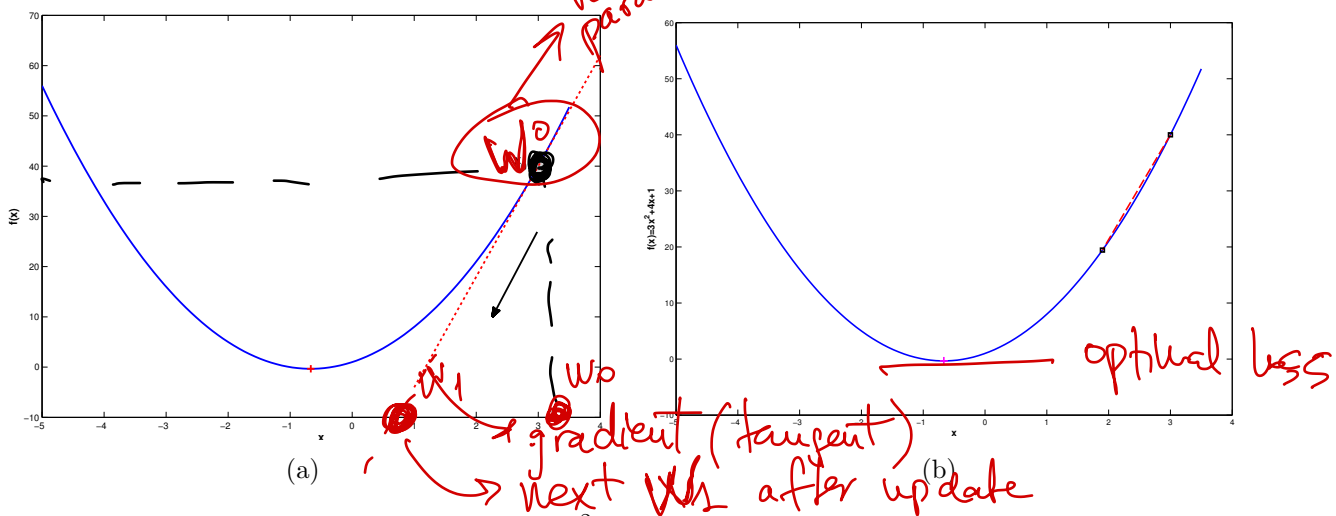
and we want to find the minimizer x^* such that $\forall x \; f(x^*) \leq f(x)$.

Since, this is a simple function we can find the minimizer analytically by taking the derivative of the function and equating it equal to zero:

$$f'(x) = 6x + 4 = 0$$
$$\Rightarrow x^* = -\frac{2}{3}$$

As the second derivative of the function $0 < f''(x) = 6$ we know that x^* is a minimizer of $f(x)$. Figure-3a shows a plot of the function.

Assume that we have x_0 as out starting point (Figure-3b), and we want to reach the minimum of the function. The derivative gives the slope of the tangent line to the function at x_0, which is positive in this case. So moving in the direction of the negative slope we will move to a point x_1 such that $f(x_1) \leq f(x_0)$ (Figure-3b).



(a) (b)

Figure 3: (a) A plot of the function $f(x) = 3x^2 + 4x + 1$ and the tangent to the function at $x_0 = 3$, if we move in the direction as shown we will be getting closer to the function minimum. (b) shows the next point x_1 that we reach by moving in the shown direction, which is closer to the minimum.

3.1 Gradient Vector

Most of the problems that we are working with involve more than one parameter, that need to be estimated by minimizing the cost function. For example the cost function defined in Equation-4 defines a mapping: $J(\mathrm{w}) : \mathbb{R}^m \rightarrow \mathbb{R}$. To minimize such function we need to define the gradient vector which is given as:

$$\nabla_w f = \begin{bmatrix} \frac{\delta f}{\delta w_1} & \frac{\delta f}{\delta w_2} & \cdots & \frac{\delta f}{\delta w_m} \end{bmatrix}^T$$

where, the i^{th} element corresponds to the partial derivative of f w.r.t. w_i. The gradient vector specifies the direction of maximum increase in f at a given point.

3.2 Gradient Descent for Function Minimization

To minimize a function w.r.t. multiple variables, we can use the gradient vector to find the direction of maximum increase, and then move in the opposite direction. Algorithm-1 outlines the steps of the gradient descent algorithm.

3.3 Learning Rate

The gradient specifies the direction that we should take to minimize the function, but it is not clear how much we should move in that direction. The learning rate (η) (also known as step size) specifies the distance we should move to calculate the next point. The algorithm in general is sensitive to the learning rate, if the value is too small it would take a long time to converge, on the other hand if the value is too large we can overshoot the minimum value and oscillate between intermediate values. Figure-4, shows the case when the learning rate is too large.

3.4 When to stop?

We need a termination condition for the gradient descent. For this purpose a number of different heuristics can be employed:

- **Maximum Number of Iterations:** Run the gradient descent for as long as we can afford to run.
- **Threshold on function change:** We can check for the decrease in function values between iterations, and if the change in values is lower than a suitable threshold τ then we can stop the algorithm i.e., $f(x_{i+1}) - f(x_i) < \tau$.

Data: Starting point:x_0, Learning Rate: η
Result: Minimizer: x^*
while *convergence criterion not satisfied* **do**
  estimate the gradient at x_i: $\nabla f(x_i)$;
  calculate the next point: $x_{i+1} = x_i - \eta \nabla f(x_i)$
end

Algorithm 1: The gradient descent algorithm for minimizing a function.

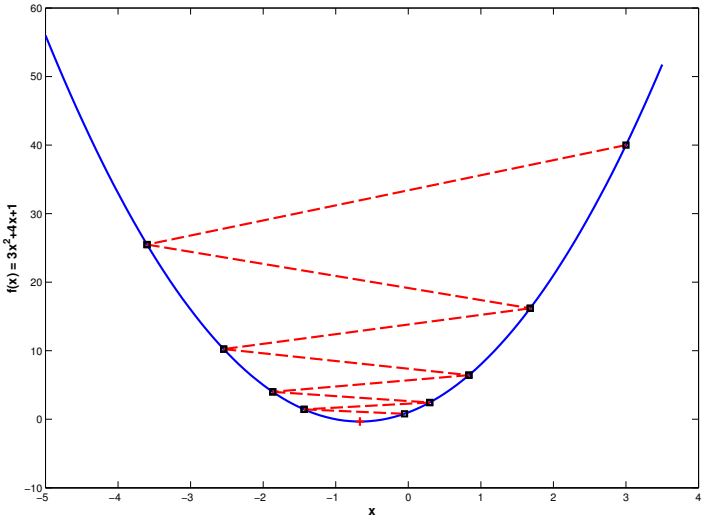


Figure 4: Behavior of gradient descent when the learning rate is too large. It can be seen that the algorithm keeps oscillating about the minimum value.

- **Thresholding the gradient:** At the optimal point (the minimizer), the gradient should theoretically be equal to zero. In most practical implementation it will not be possible to achieve achieve this, in such a case we can define a tolerance parameter ϵ and stop when $||\nabla f|| < \epsilon$.

4 Gradient Descent for Linear Regression

Recall, that for linear regression we are trying to minimize the cost function:

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}(\mathbf{w}^T\mathbf{x}_i - y_i)^2$$

square loss : minimize error

where, we have multiplied the function by 0.5 which only scales the function and does not affect where the minimum occurs. We need to calculate the gradient $\nabla_w J$. To this end we are going to calculate the partial derivatives of $J(w)$ w.r.t. the individual parameters w_k:

$$\frac{\delta}{\delta w_k}J = \frac{1}{2}\frac{\delta}{\delta w_k}\sum_{i=1}^{N}(\mathbf{w}^T\mathbf{x}_i - y_i)^2$$

$$= \sum_{i=1}^{N}(\mathbf{w}^T\mathbf{x}_i - y_i)\,\frac{\delta}{\delta w_k}\sum_{i=1}^{N}(\mathbf{w}^T\mathbf{x}_i - y_i)$$

$$= \sum_{i=1}^{N}(\mathbf{w}^T\mathbf{x}_i - y_i)(x_{ik})$$

to know how to update w

gradients $\frac{\partial J}{\partial w}$ are based on $h_w(x) = \langle w \cdot x \rangle$ model (h_w = linear)

based on these calculation we can define the gradient descent update rule for w_k as:

G.D. update w update:

$$w_k^{i+1} = w_k^i - \eta\sum_{i=1}^{N}((\mathbf{w}^i)^T\mathbf{x}_i - y_i)(x_{ik}) \quad \forall k \in \{1, 2, \ldots, m\}$$

Data: Design Matrix: $X \in \mathbb{R}^{Nxm}$, Label Vector: $\mathbf{y} \in \mathbb{R}^N$
Initial weights:$\mathbf{w_0} \in \mathbb{R}^m$, Learning Rate: η
Result: Least squares minimizer: $\mathbf{w}^*$
while *convergence criterion not satisfied* **do**
  estimate the gradient at $\mathbf{w}_i$: $\nabla J(\mathbf{w}_i)$;
  updatet: $\mathbf{w_{i+1}} = \mathbf{w_i} - \eta \nabla J(\mathbf{w}_i)$
end

Algorithm 2: The gradient descent algorithm for finding the least squares estimator (LSE) for the regression function.

Instead of updating each weight individually we can formulate the gradient vector and define a vector update rule that updates all the weights simultaneously,

$$\nabla_w J = \sum_{i=1}^{N} (\mathbf{w}^T \mathbf{x}_i - y_i)(\mathbf{x}_i)$$

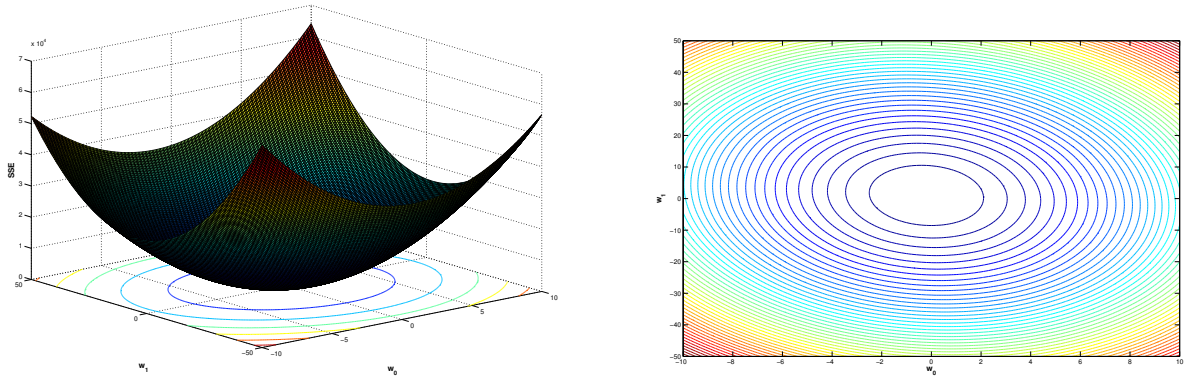
the update rule is:

$$\mathbf{w}^{i+1} = \mathbf{w}^i - \eta \sum_{i=1}^{N} ((\mathbf{w}^i)^T \mathbf{x}_i - y_i)(\mathbf{x}_i) \tag{5}$$

Algorithm-2 outlines the algorithm for learning the optimal weight vector for linear regression. It should be noted here that the optimality of the weight vector is based on the SSE criterion, and therefore the optimal weight vector is known as Least Squares Estimator (LSE).

Example: For our initial example we need to calculate the least squares solution for $\mathbf{w} = \begin{bmatrix} w_0 \, w_1 \end{bmatrix}^T$. For the housing dataset, using only the average number of rooms in the dwelling to predict the house price, the sum of squares error (SSE) is shown in Figure-5.

An example run of the gradient descent algorithm on the housing data is shown in Figure-6.



(a) (b)

Figure 5: (a) A plot of the sum of squares cost function for the housing dataset, where we are trying to predict the price of the house based only on the average number of rooms in the house. (a) shows a three-dimensional plot of the cost function, while (b) shows the contours of this plot, where red indicates higher values of SSE while blue indicates lower SSE.

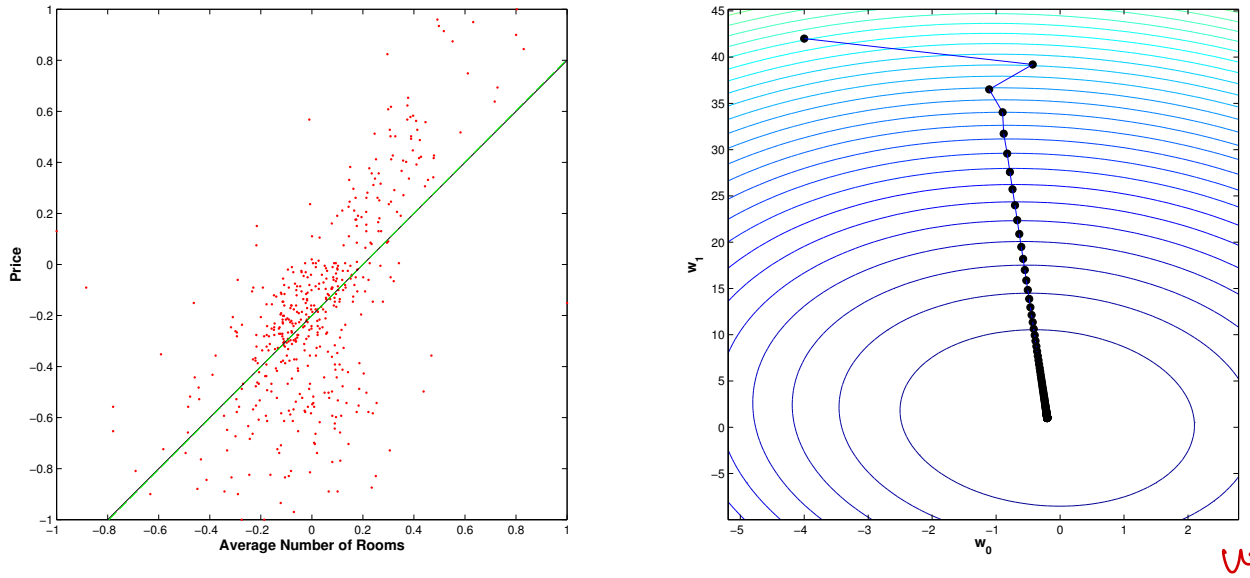


Figure 6: A run of the gradient descent algorithm on the housing dataset from our example. (b) shows the convergence trajectory taken by the gradient descent algorithm, and (a) shows the optimal regression line corresponding to $\mathbf{w}^*$ estimated from the gradient descent algorithm.

5 Least squares via normal equations

Given training data $(\mathbf{x}_i, y_i)$ for $i = 1, 2, ..., N$, with m input features, arranged in the design matrix X and the label vector $\mathbf{y}$.

$$X = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1m} \\ x_{21} & x_{22} & \dots & x_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N1} & x_{N2} & \dots & x_{Nm} \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

Then the error array associated with our regressor $f_{\mathbf{w}}(\mathbf{x}) = \sum_{k=1}^{m} w_k x_k$ is:

$$E = \begin{bmatrix} f_{\mathbf{w}}(\mathbf{x}_1) - y_1 \\ \dots \\ f_{\mathbf{w}}(\mathbf{x}_N) - y_N \end{bmatrix} = \begin{bmatrix} \mathbf{x}_1^T \mathbf{w} \\ \dots \\ \mathbf{x}_N^T \mathbf{w} \end{bmatrix} - \begin{bmatrix} y_1 \\ \dots \\ y_N \end{bmatrix} = X\mathbf{w} - \mathbf{y}$$

(we used $w = (w_0, w_1, ...w_m)^T$ as a column vector). We can now write the sum of squares error as

$$J(w) = \frac{1}{2} \sum_{i=1}^{N} (f_w(\mathbf{x}_i) - y_i)^2 = \frac{1}{2} E^T E = \frac{1}{2} (X\mathbf{w} - \mathbf{y})^T (X\mathbf{w} - \mathbf{y})$$

Then,

$$
\begin{aligned}
\nabla_w J(\mathbf{w}) &= \nabla_w \frac{1}{2} E^T E = \nabla_w \frac{1}{2} (X\mathbf{w} - \mathbf{y})^T (X\mathbf{w} - \mathbf{y}) \\
&= \frac{1}{2} \nabla_w (\mathbf{w}^T X^T X\mathbf{w} - \mathbf{w}^T X^T \mathbf{y} - \mathbf{y}^T X\mathbf{w} + \mathbf{y}^T \mathbf{y}) \\
&= \frac{1}{2} \nabla_w (\mathbf{w}^T X^T X\mathbf{w} - \mathbf{w}^T X^T \mathbf{y} - \mathbf{w}^T X^T \mathbf{y} + \mathbf{y}^T \mathbf{y}) \\
&= \frac{1}{2} \nabla_w (\mathbf{w}^T X^T X\mathbf{w} - 2\mathbf{w}^T X^T \mathbf{y} + \mathbf{y}^T \mathbf{y}) \\
&= \frac{1}{2} 2(X^T X\mathbf{w} - X^T \mathbf{y}) \\
&= X^T X\mathbf{w} - X^T \mathbf{y}
\end{aligned}
$$

where, in the third step we have $(AB)^T = B^T A^T$.

Since we are trying to minimize J, a convex function, a sure way to find w that minimizes J is to set its derivative to zero. In doing so we obtain

$$X^T X\mathbf{w} = X^T \mathbf{y} \text{ or } \mathbf{w} = (X^T X)^{-1} X^T \mathbf{y}$$

This is the exact $\mathbf{w}$ that minimizes the sum of squares error.