CS 6140: Machine Learning
College of Computer and Information Science
Northeastern University
Module2 Lesson2
Instructor: Bilal Ahmed
Scribe: Bilal Ahmed & Virgil Pavlu

Gradient Descent For Linear Regression[1]

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:

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

1 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-1a shows a plot of the function.

Assume that we have x_0 as out starting point (Figure-1b), 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-1b).

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

1.1 Gradient Vector

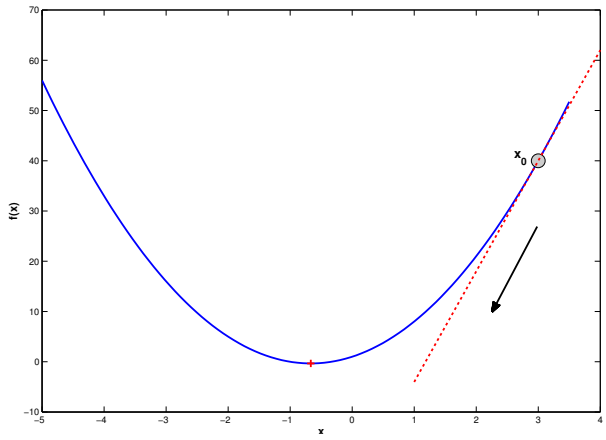
Most of the problems that we are working with involve more than one parameter, that need to be estimated by minimizing the cost function. For example the cost function defined in Equation-1 defines a mapping: $J(\mathrm{w}) : \mathbb{R}^m \rightarrow \mathbb{R}$. To minimize such function we need to define the gradient vector which is given as:

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

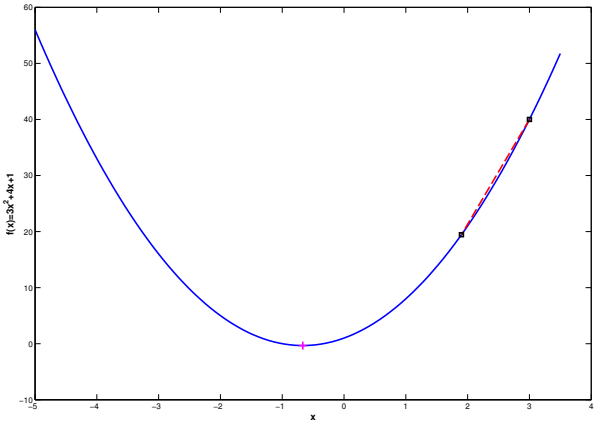
where, the i^{th} element corresponds to the partial derivative of f w.r.t. w_i. The gradient vector specifies the direction of maximum increase in f at a given point.

1.2 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.



(a)



(b)

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.

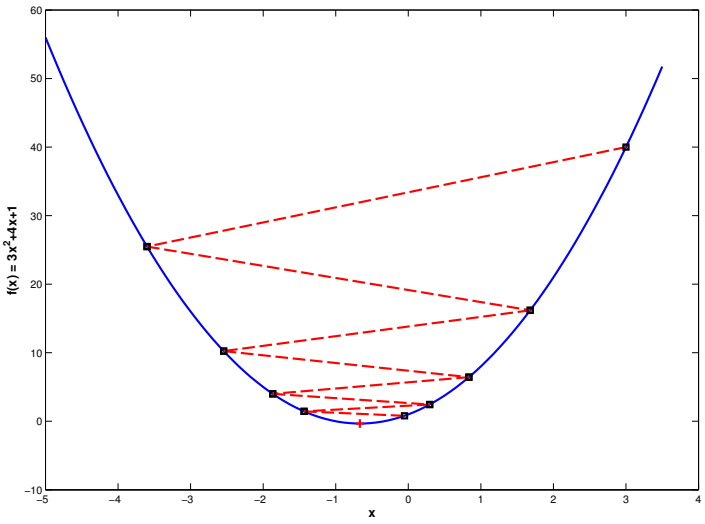


Figure 2: Behavior of gradient descent when the learning rate is too large. It can be seen that the algorithm keeps oscillating about the minimum value.

1.3 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-2, shows the case when the learning rate is too large.

1.4 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., $f(x_{i+1}) - f(x_i) < \tau$.
- **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 $||\nabla f|| < \epsilon$.

2 Gradient Descent for Linear Regression

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

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

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:

$$\begin{aligned}\frac{\delta}{\delta w_k}J &= \frac{1}{2}\,\frac{\delta}{\delta w_k}\sum_{i=1}^{N}(\mathbf{w}^T\mathbf{x}_i - y_i)^2 \\ &= \sum_{i=1}^{N}(\mathbf{w}^T\mathbf{x}_i - y_i)\,\frac{\delta}{\delta w_k}\sum_{i=1}^{N}(\mathbf{w}^T\mathbf{x}_i - y_i) \\ &= \sum_{i=1}^{N}(\mathbf{w}^T\mathbf{x}_i - y_i)(x_{ik})\end{aligned}$$

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

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

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:

$$\mathbf{w}^{i+1} = \mathbf{w}^i - \eta\,\sum_{i=1}^{N}((\mathbf{w}^i)^T\mathbf{x}_i - y_i)(\mathbf{x}_i) \tag{2}$$

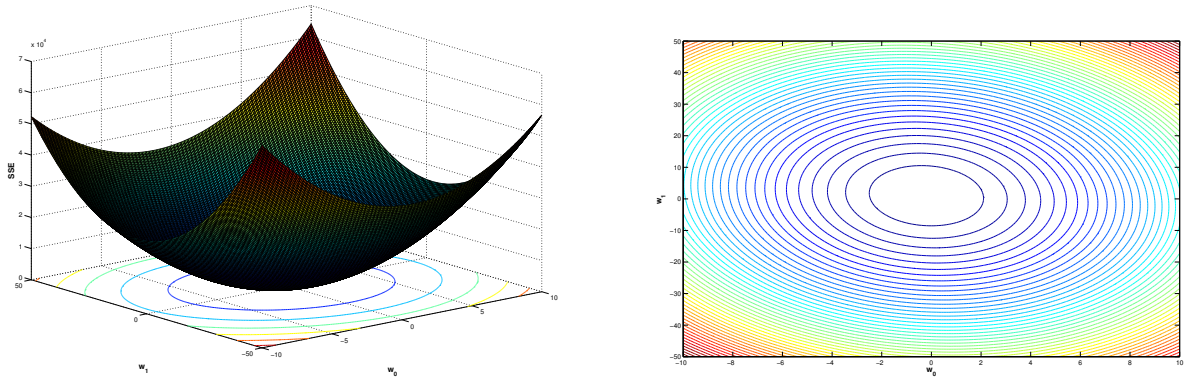
Algorithm-2 outlines the algorithm for learning the optimal weight vector for linear regression. It should be noted here that the optimality of the weight vector is based on the SSE criterion, and therefore the optimal weight vector is known as Least Squares Estimator (LSE).

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-3.

An example run of the gradient descent algorithm on the housing data is shown in Figure-4.

Data: Design Matrix: $X \in \mathbb{R}^{N \times m}$, 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.



(a) (b)

Figure 3: (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.

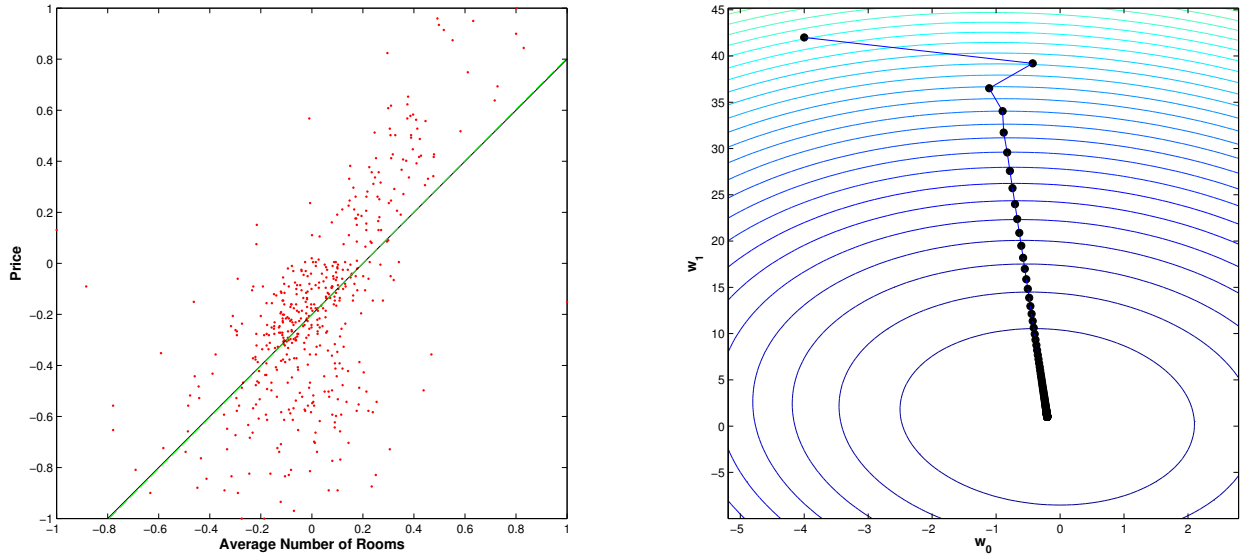


Figure 4: A run of the gradient descent algorithm on the housing dataset from our example. (b) shows the convergence trajectory taken by the gradient descent algorithm, and (a) shows the optimal regression line corresponding to $\mathbf{w}^*$ estimated from the gradient descent algorithm.