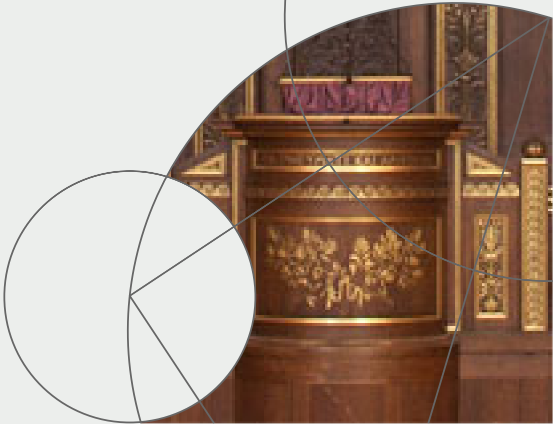
STEVENS INSTITUTE OF TECHNOLOGY

Machine Learning Fundamentals: Gradient Descent

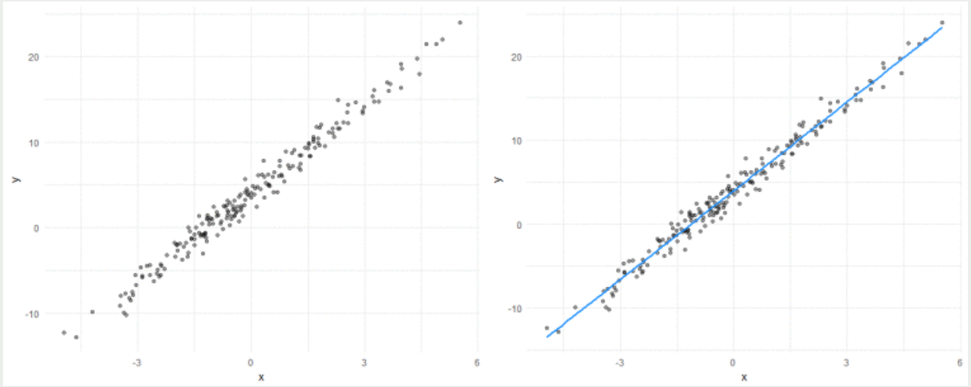
Rensheng Wang,
https://sit.instructure.com/courses/34886

September 4, 2019

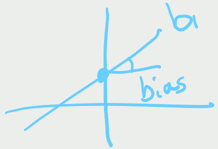


Linear Regression

- Use a simple linear regression model to explain the machine learning (ML) fundamental: gradient descent.
- Below is an example to show the linear regression fit for data variable X and Y.



Linear Regression



- In the case of the simple linear regression

$$y \sim b_0 + b_1 * x$$

where x is one column/variable and the model “learns” two parameters:

  - b_0: the bias
  - b_1: the slope

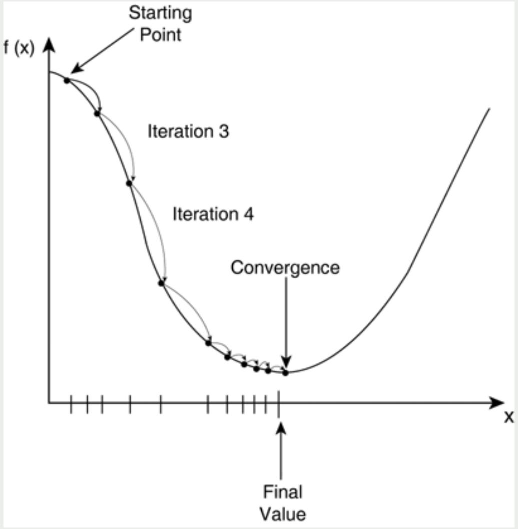
- The bias is the level of y when x is 0 and the slope is the rate of predicted increase or decrease in y for each unit increase in x. Both parameters are scalars (single values).
- Once the model learns these parameters they can be used to compute estimated values of y given new values of x. In other words, you can use these learned parameters to predict values of y when you dont know what y is $--$ a predictive model!

Learning parameters: Cost functions

- There are several ways to learn the parameters of a LR model, I will focus on the approach that best illustrates statistical learning; minimizing a cost function.
- A cost function – it helps the learner to correct / change behavior to minimize mistakes
- A cost function is a measure of how wrong the model is in terms of its ability to estimate the relationship between x and y.
- This is typically expressed as a difference or distance between the predicted value and the actual value.
- The cost function (you may also see this referred to as loss or error.) can be estimated by iteratively running the model to compare estimated predictions against ground truth the known values of y.
- The objective of a ML model, therefore, is to find parameters, weights or a structure that minimizes the cost function.

Minimizing the cost function: Gradient descent

- Gradient descent is an efficient optimization algorithm that attempts to find a local or global minima of a function.

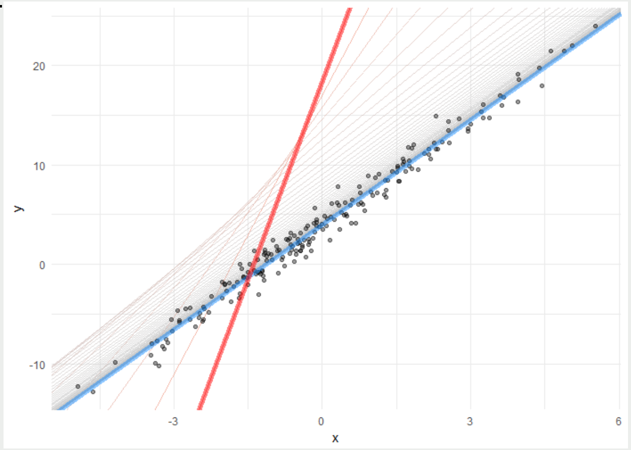


Minimizing the cost function: Gradient descent

- Gradient descent enables a model to learn the gradient or direction that the model should take in order to reduce errors (differences between actual y and predicted y).
- Direction in the simple linear regression example refers to how the model parameters b_0 and b_1 should be tweaked or corrected to further reduce the cost function.
- As the model iterates, it gradually converges towards a minimum where further tweaks to the parameters produce little or zero changes in the loss also referred to as convergence.
- At this point the model has optimized the weights such that they minimize the cost function.

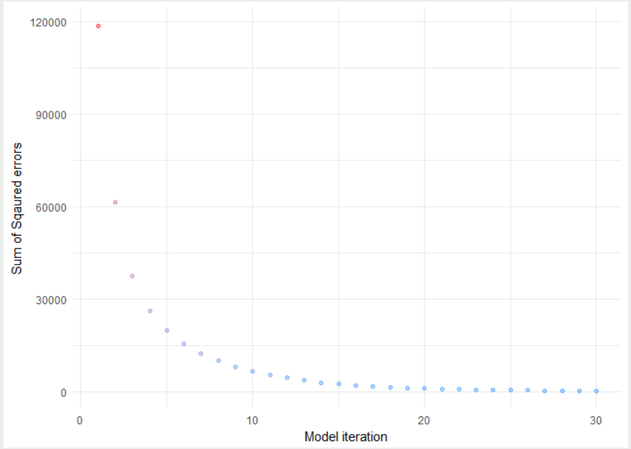
Minimizing the cost function: Gradient descent

- We define the MSE(mean square errors) as the cost function. It is simply the mean of the squared differences between predicted y and actual y (i.e. the residuals).
- We randomly assign the estimated values of the bias and the slope, which is the red line shown in the plot. The line gradually moves to the blue line as the gradient descent minimizing the MSE.



Minimizing the cost function: Gradient descent

- We can also visualize the decrease in the sum of squared errors across iterations of the model. This takes a steep decline in the early iterations before converging and stabilizing.



Gradient

$x \longrightarrow f(x) \rightarrow$ the loss fn

- In order to perform the gradient descent, we need the function is a differentiable function. This is why the activation functions are always differentiable functions.
- For scalar x, gradient is denoted as $f'(x)$. For multi-dimensional variable $\mathbf{x}$, we use $\nabla f(\mathbf{x})$.

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}, \quad \nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} \\ \frac{\partial f(\mathbf{x})}{\partial x_2} \\ \vdots \\ \frac{\partial f(\mathbf{x})}{\partial x_d} \end{bmatrix}$$

Gradient

- For example, we have $\mathbf{x}$ with 2-dimensional,

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \quad \mathbf{a} = \begin{bmatrix} 10 \\ 1 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 5 & 4 \\ 3 & 2 \end{bmatrix}$$

- When $f(\mathbf{x}) = \mathbf{a}^T\mathbf{x} + 1 = [10, \ 1] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + 1 = 10x_1 + x_2 + 1$

- The gradient

$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} \\ \frac{\partial f(\mathbf{x})}{\partial x_2} \end{bmatrix} = \begin{bmatrix} \frac{\partial(10x_1+x_2+1)}{\partial x_1} \\ \frac{\partial(10x_1+x_2+1)}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 10 \\ 1 \end{bmatrix}$$

Gradient

- For more complicated cases,

$$\begin{aligned} f(\mathbf{x}) =& \mathbf{x}^T\mathbf{B}\mathbf{x} + \mathbf{a}^T\mathbf{x} + 1 \\ =& [x_1, x_2]\begin{bmatrix} 5 & 4 \\ 3 & 2 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + [10, 1]\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + 1 \\ =& 5x_1^2 + 2x_2^2 + 7x_1x_2 + 10x_1 + x_2 + 1 \end{aligned}$$

- The gradient $\nabla f(\mathbf{x})$

$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} \\ \frac{\partial f(\mathbf{x})}{\partial x_2} \end{bmatrix}$$

$$= \begin{bmatrix} \frac{\partial(5x_1^2+2x_2^2+7x_1x_2+10x_1+x_2+1)}{\partial x_1} \\ \frac{\partial(5x_1^2+2x_2^2+7x_1x_2+10x_1+x_2+1)}{\partial x_2} \end{bmatrix}$$

$$= \begin{bmatrix} 10x_1 + 7x_2 + 10 \\ 4x_2 + 7x_1 + 1 \end{bmatrix}$$

Gradient Descent

- Gradient Descent is a method to iteratively update the parameters via minimizing the cost function.
- For each iteration, the updated parameter set $\mathbf{x}^{(t+1)}$

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \gamma \nabla f(\mathbf{x}^{(t)})$$

  where γ is the learning rate and $\mathbf{x}^{(t)}$ is the previous step result.
- The gradient tells you the direction for updating the solutions and the learning rate tells you how much you should move toward the gradient direction.

Gradient Descent (Example 1)

- Example 1:

$$f(x) = x^2 - 10x + 1$$

$$\nabla f(\mathbf{x}) = f'(x) = 2x - 10$$

- Use the gradient descent to minimize the cost function $f(x)$:

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \gamma \nabla f(\mathbf{x}^{(t)})$$

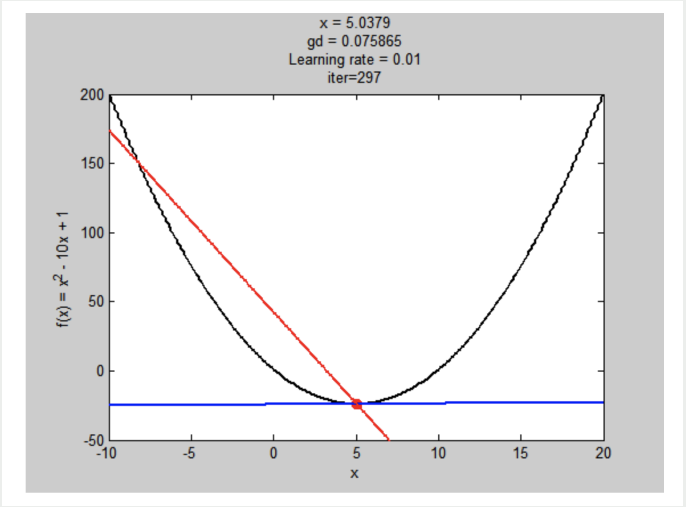
$$\Rightarrow \; x^{(t+1)} = x^{(t)} - \gamma(2x^{(t)} - 10) = (1 - 2\gamma)x^{(t)} + 10\gamma$$

$$\Rightarrow \; x^{(t+1)} = (1 - 2\gamma)x^{(t)} + 10\gamma$$

- Let us examine how different learning rates γ will affect the gradient descent.

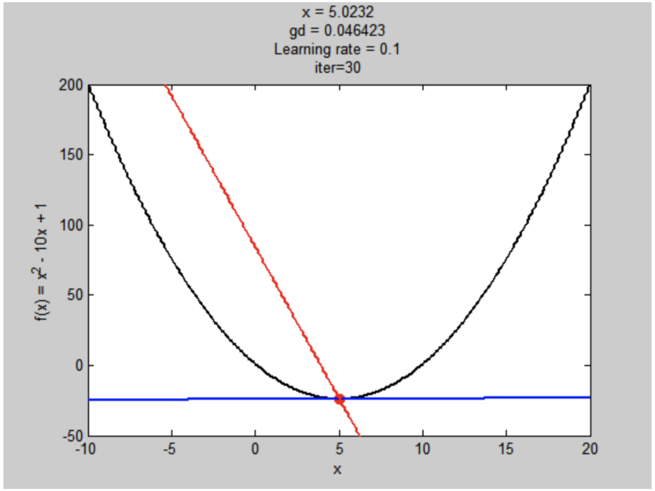
Gradient Descent (Example 1)

- Learning rate $\gamma = 0.01$



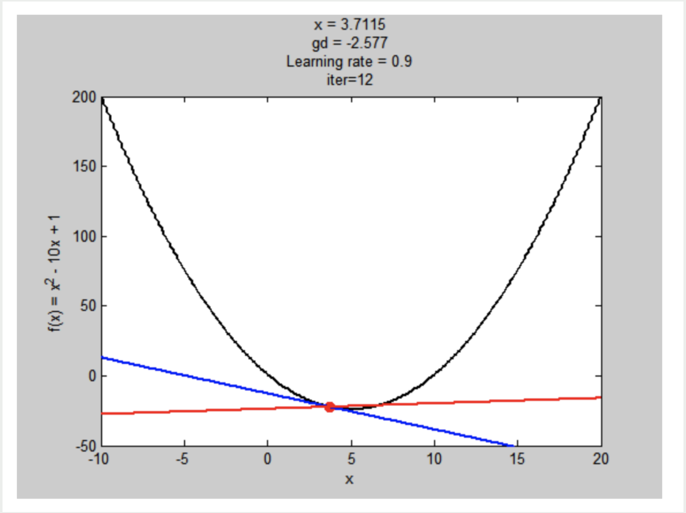
Gradient Descent (Example 1)

- Learning rate $\gamma = 0.1$



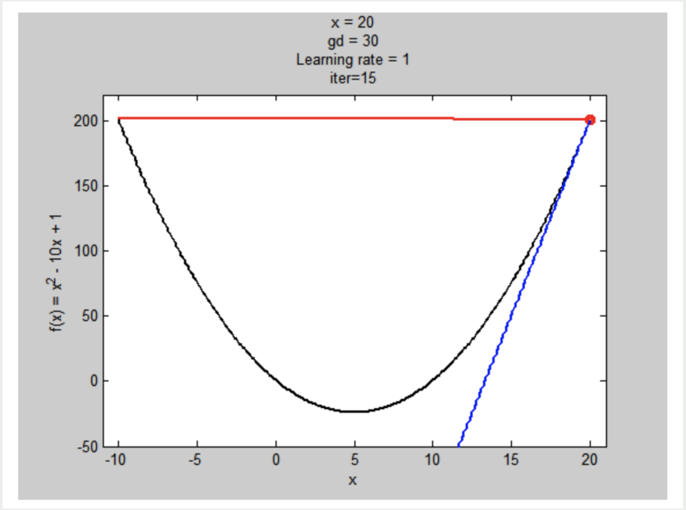
Gradient Descent (Example 1)

- Learning rate $\gamma = 0.9$



Gradient Descent (Example 1)

- Learning rate $\gamma = 1$



Gradient Descent (Example 2)

- Example 2:

$$f(x) = x^4 - 50x^3 - x + 1$$

$$\nabla f(\mathbf{x}) = f'(x) = 4x^3 - 150x^2 - 1$$

- Use the gradient descent to minimize the cost function $f(x)$:

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \gamma \nabla f(\mathbf{x}^{(t)})$$

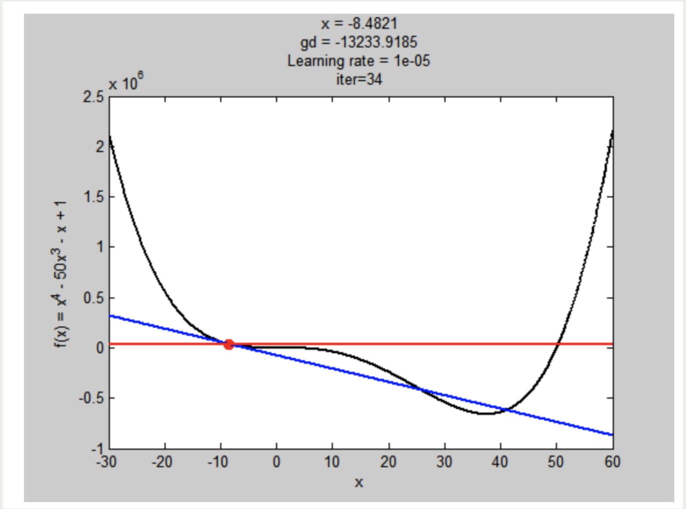
$$\Rightarrow \; x^{(t+1)} = x^{(t)} - \gamma(4{x^{(t)}}^3 - 150{x^{(t)}}^2 - 1)$$

$$\Rightarrow \; x^{(t+1)} = -4\gamma {x^{(t)}}^3 + 150\gamma {x^{(t)}}^2 + x^{(t)} + \gamma$$

- Let us examine how different learning rates γ will affect the gradient descent.

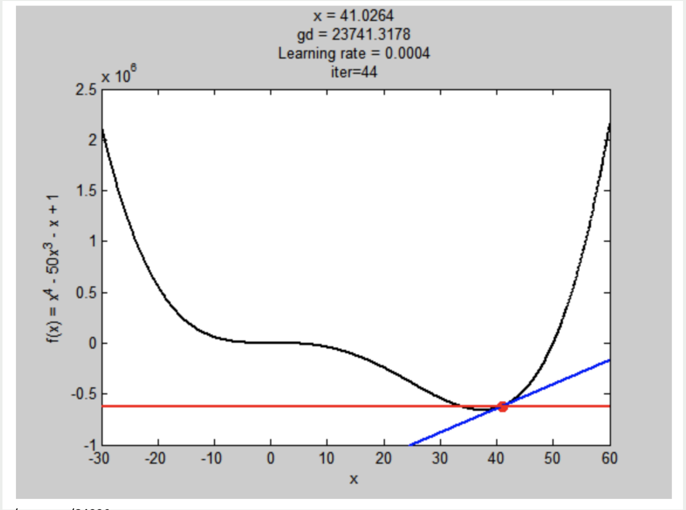
Gradient Descent (Example 2)

- Learning rate $\gamma = 0.00001$. Initial value is bad and the solution is a local minimum.



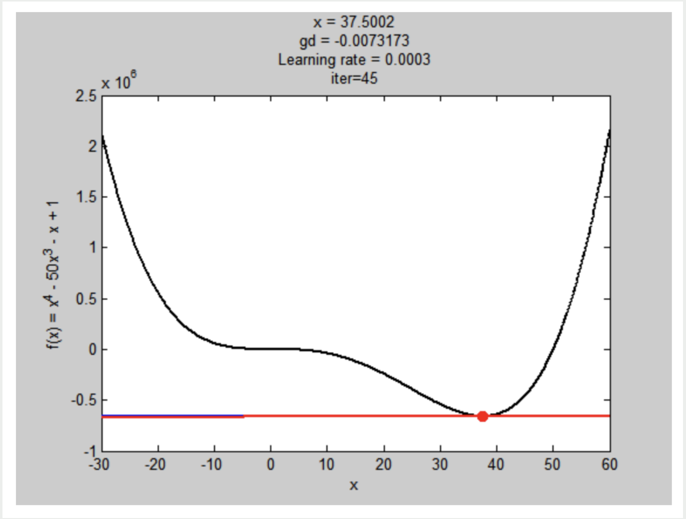
Gradient Descent (Example 2)

- Learning rate $\gamma = 0.0004$. A larger learning rate can avoid the local minimum but when it approaches the global minimum, it cannot converge to the optimum solution due to a larger learning step.



Gradient Descent (Example 2)

- Learning rate $\gamma = 0.0003$. A larger learning rate can avoid the local minimum and it can also reach the global minimum.



Stochastic Gradient Descent

- We use a simple example - linear regression, to demonstrate the difference between the stochastic gradient descent vs. the vanilla gradient descent
- For linear regression $y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots$, we have the cost function

$$J(\boldsymbol{\theta}) = \frac{1}{2m}\sum_{i=1}^{m}(\hat{y}^{(i)} - y^{(i)})^2$$

where $\boldsymbol{\theta}$ contains the set of unknown parameters$(\theta_0, \theta_1, \theta_2, \cdots)$, m is the total number of samples, $y^{(i)}$ is the i-th sample, and $\hat{y}^{(i)}$ is the predicted i-th sample of $y^{(i)}$.

- The gradients function $\nabla J(\boldsymbol{\theta}) = \begin{bmatrix} \frac{\partial}{\partial\theta_0}J(\boldsymbol{\theta}) \\ \frac{\partial}{\partial\theta_1}J(\boldsymbol{\theta}) \\ \vdots \end{bmatrix}$, where

$$\frac{\partial}{\partial\theta_j}J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}^{(i)} - y^{(i)}) \cdot x_j^{(i)}$$

Stochastic Gradient Descent (SGD)

- We use a simple example - linear regression, to demonstrate the difference between the stochastic gradient descent vs. the vanilla gradient descent
- For vanilla gradient descent, the unknown parameters θ_j is updated as

$$\theta_j \;\Leftarrow\; \theta_j - \gamma \frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta}) = \theta_j - \gamma \frac{1}{m} \sum_{i=1}^{m} (\hat{y}^{(i)} - y^{(i)}) \cdot x_j^{(i)}$$

- For stochastic gradient descent, we use the cost gradient of 1 randomly selected sample at each iteration, instead of using the sum of the cost gradient of ALL samples.

```
for i in range(m) :
    $\theta_j \;\Leftarrow\; \theta_j - \gamma \left[ \left( \hat{y}^{(i)} - y^{(i)} \right) \cdot x_j^{(i)} \right]$
```

Stochastic Gradient Descent (SGD)

- In stochastic gradient descent, before each iteration loop, you need to randomly shuffle the training examples.
- In SGD, because its using only one example at a time, its path to the minima is noisier (more random) than that of the vanilla batch gradient. But its ok as we are indifferent to the path, as long as it gives us the minimum AND the shorter training time.
- Mini-batch gradient descent uses n data points (instead of 1 sample in SGD) at each iteration.