Linear and Polynomial Regression

Linear Regression

A linear model makes a prediction by simply computing a weighted sum of the input feature, plus a constant called the bias term (intercept term).

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

ŷ is the predicted value

n is the number of features

xi is the ith feature value

θj is the jth model parameters (including the bias term θ0 and the feature weights, θ1, θ2, …, θn).

Normal equation

To find the value of θ that minimizes the cost function, there is a closed-form solution.

$$\widehat{\boldsymbol{\theta}} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1} \quad \mathbf{X}^T \quad \mathbf{y}$$

where

θ is the value that minimizes the cost function

X is the designed matrix containing the input features of the training data

y is the vector of the target values

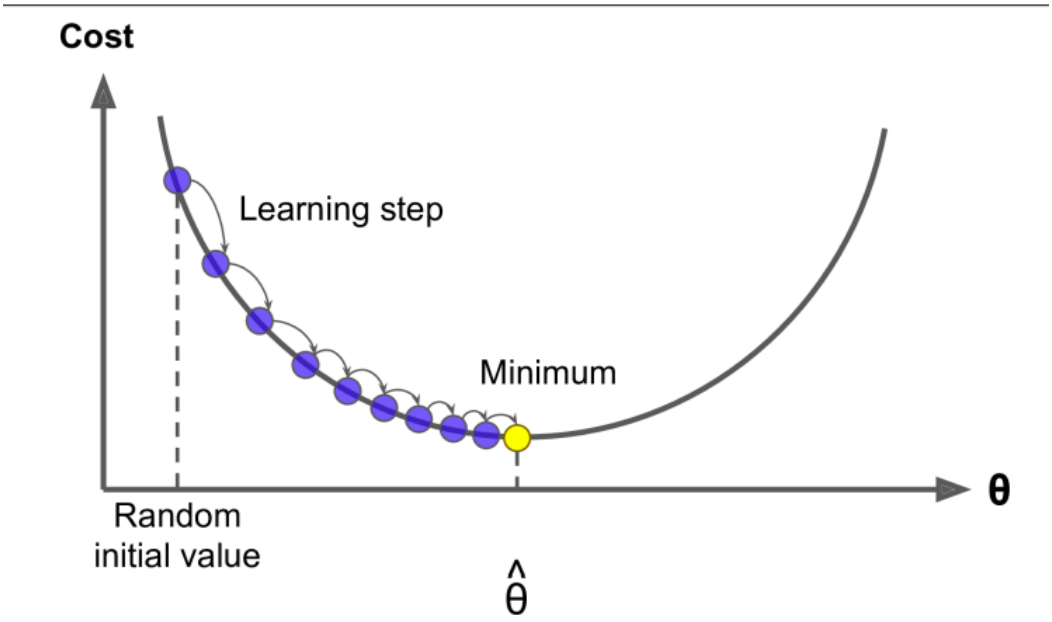
Computational Complexity

The normal equation computes the inverse of X^T*X, which is an (n+1) * (n+1) matrix. This takes typically about O(n^2.4) to O(n^3), i.e., it's very expensive.

Gradient Descent

Another optimization algorithm capable of finding optimal solutions. The general idea of GD is to adjust the parameters iteratively in order to minimize a cost function.

It measures the local gradient of the error function with regard to the parameter vector θ and it goes in the direction of descending gradient. Once the gradient is zero, you have reached a minimum.



An important parameter is the size of the steps, determined by the learning rate hyperparameter. If the learning rate is too small, then the algorithm will have to go through many iterations to converge, which will take a long time.

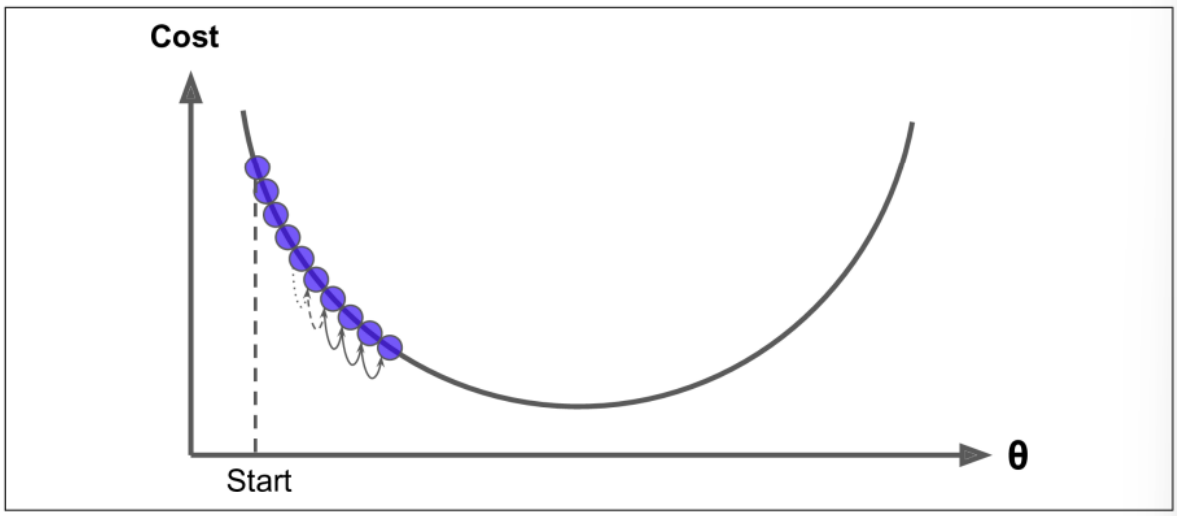


Figure 4-4. Learning rate too small

On the other hand, if the learning rate is too high, you might jump across the valley and end up on the other side, possibly even higher up than you were before. This might make the algorithm diverge.

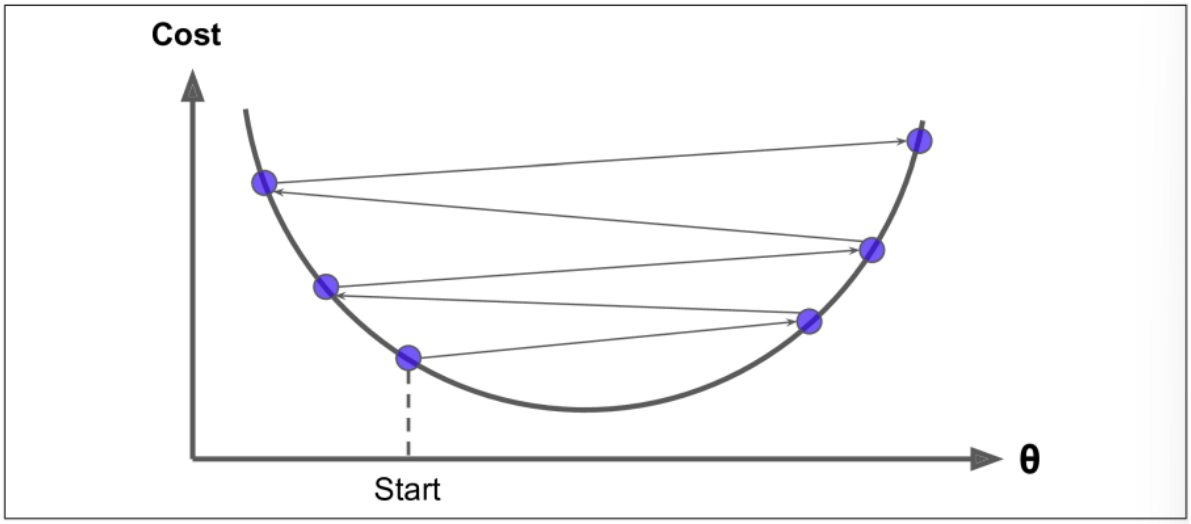


Figure 4-5. Learning rate too large

The MSE cost function for a Linear Regression model happens to be a convex function. This implies that there are no local minima, just one global minimum. It's also a continuous function with a slope that never changes abruptly. These two facts have great consequences. GD is guaranteed to approach arbitrarily close to the global minimum.

Pros:

Simple and fast for strongly convex problems

Cons:

Slow, requires many iterations and small alpha in many realistic cases.

Multivariate Linear Regression

Model the relationship between multiple independent variables and a dependent variable.

y = β0 + β1*X1 + β2*X2 + ... + βp*Xp

Polynomial Regression

Observation: A polynomial of degree 3 could handle the up-down-up scenario
Model: $y = \beta_1 + \beta_2 X + \beta_3 X^2 + \beta_4 X^3$

Vectorised Approach Model $y = X\beta$

$$X = \begin{bmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^2 & x_2^3 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & x_2^2 & x_n^3 \end{bmatrix}, \quad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, \quad \beta = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \end{bmatrix}$$

Train and Test error

Training error: Compute MSE with Xtrain, ytrain

Test error: Compute MSE with Xtest, ytest

Variance and Bias, Over and Underfitting

The reducible error for model f(x) can further be divided into two parts

Reducible Error = Variance + Bias

Variance: no systematic error but oversensitive to training data

Bias: Systematic error due to a much too simple model, can't capture all features of the dataset