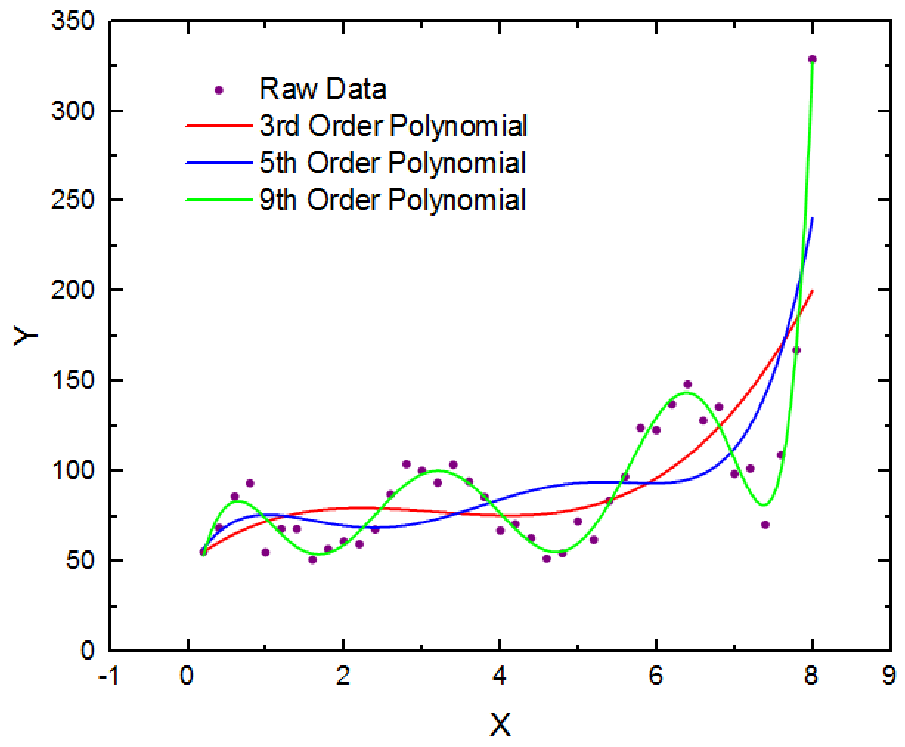
ECE 4252/8803: Fundamentals of Machine Learning (FunML)
Fall 2024

Lecture 8: Polynomial Regression



Recap

Lecture 7

Simple Linear Regression Model

General Linear Regression Model

Cost Function

Finding Model Coefficients Using the Normal Equation

Linear Regression

General Linear Regression Model

- A linear regression *predictor* aims to estimate this relationship and find *estimated* $\hat{\boldsymbol{y}}$ based on *estimated* coefficients $\hat{\boldsymbol{\theta}}$.
  This is expressed in matrix form as follows

$$\hat{\boldsymbol{y}} = \boldsymbol{X}\hat{\boldsymbol{\theta}} \quad \text{or} \quad \boldsymbol{y} = \boldsymbol{X}\hat{\boldsymbol{\theta}} + \boxed{\boldsymbol{\varepsilon}},$$

Residual error

where

$$\hat{\boldsymbol{y}} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_N \end{bmatrix}, \boldsymbol{x_i} = \begin{bmatrix} 1 \\ x_{i1} \\ \vdots \\ x_{iP} \end{bmatrix}, \boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1^T \\ \boldsymbol{x}_2^T \\ \vdots \\ \boldsymbol{x}_N^T \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & \cdots & x_{1P} \\ 1 & x_{21} & \cdots & x_{2P} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N1} & \cdots & x_{NP} \end{bmatrix}, \hat{\boldsymbol{\theta}} = \begin{bmatrix} \hat{\theta}_0 \\ \hat{\theta}_1 \\ \vdots \\ \hat{\theta}_P \end{bmatrix}, \boldsymbol{\varepsilon} = \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_N \end{bmatrix}$$

- Obviously, estimated target vector $\hat{\boldsymbol{y}}$ is a direct result of estimated coefficient vector $\hat{\boldsymbol{\theta}}$. Therefore, the objective in Linear Regression is to find the optimal $\hat{\boldsymbol{\theta}}$

General Linear Regression Model

Least Squares Loss Function

- Recall that a linear regression *predictor* estimates the linear relationship between observed target variable $\boldsymbol{y}$ and input variable $\boldsymbol{X}$ via *estimated* coefficients $\widehat{\boldsymbol{\theta}}$ as following:

$$\boldsymbol{y} = \boldsymbol{X}\widehat{\boldsymbol{\theta}} + \boldsymbol{\varepsilon}$$

$\boldsymbol{\varepsilon} \sim \mathcal{N}(\mu, \sigma^2)$, where μ is the mean and σ^2 is the variance; $p(\varepsilon_i) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{\varepsilon_i^2}{2\sigma^2}\right)$

$$p(y_i | \boldsymbol{x}_i; \widehat{\boldsymbol{\theta}}) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{\left(y_i - \widehat{\boldsymbol{\theta}}^T \boldsymbol{x}_i\right)^2}{2\sigma^2}\right)$$

The likelihood can be written as: $L(\widehat{\boldsymbol{\theta}}; \boldsymbol{X}, \boldsymbol{y}) = \prod_{i=1}^{N} p(y_i | \boldsymbol{x}_i; \widehat{\boldsymbol{\theta}}) = \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{\left(y_i - \widehat{\boldsymbol{\theta}}^T \boldsymbol{x}_i\right)^2}{2\sigma^2}\right)$

$$\log L(\widehat{\boldsymbol{\theta}}; \boldsymbol{X}, \boldsymbol{y}) = \sum_{i=1}^{N} \log \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{\left(y_i - \widehat{\boldsymbol{\theta}}^T \boldsymbol{x}_i\right)^2}{2\sigma^2}\right) = N \log \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{2\sigma^2} \sum_{i=1}^{N} \left(y_i - \widehat{\boldsymbol{\theta}}^T \boldsymbol{x}_i\right)^2$$

MLE (Maximize Likelihood Estimation) or Minimize the **negative log-likelihood**:

$$L(\widehat{\boldsymbol{\theta}}) = \frac{1}{N} \sum_{i=1}^{N} (\widehat{\boldsymbol{\theta}}^T \boldsymbol{x}_i - y_i)^2 \Rightarrow \widehat{\boldsymbol{\theta}} = (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{y} \text{ (Normal Equations)}$$

Materials adapted from Murphy, Kevin P. *Machine learning: a probabilistic perspective*. MIT press, 2012.

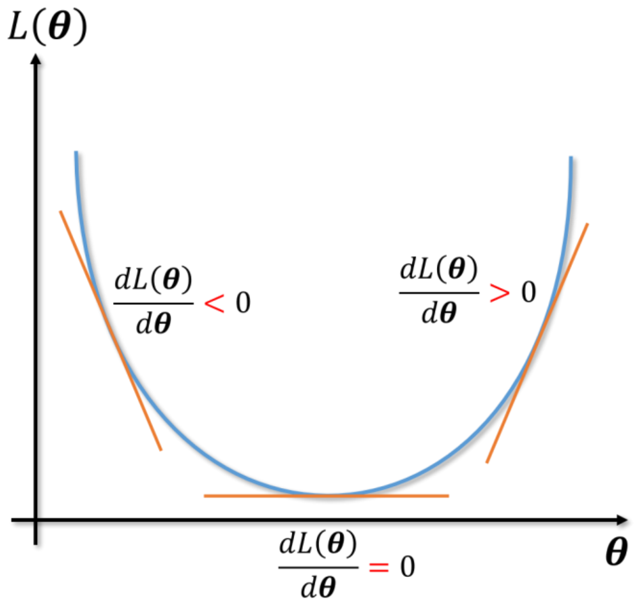
General Linear Regression Model

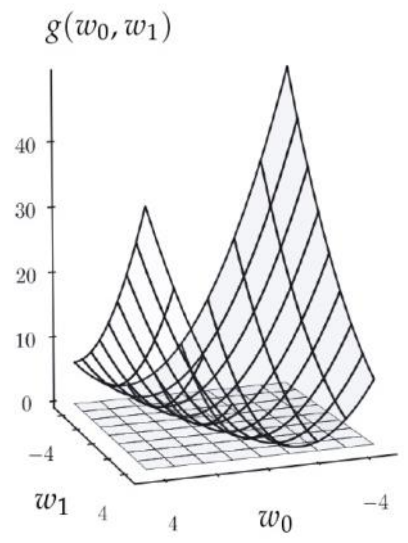
Least Squares Loss Function

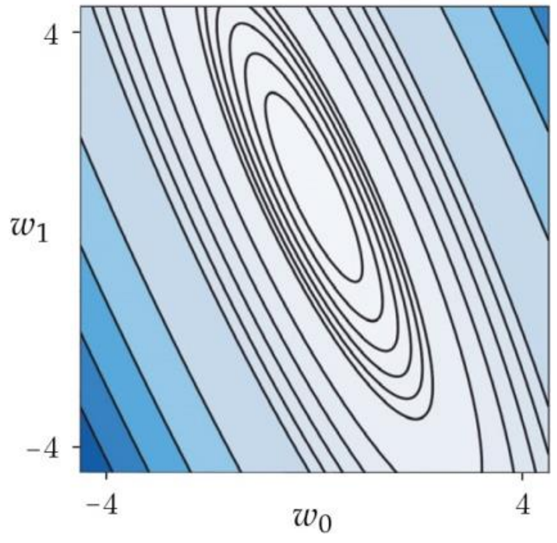
- Let $L(\widehat{\boldsymbol{\theta}})$ be the cost function that measures the difference between predictions $\widehat{\boldsymbol{y}}$ and desired outputs $\boldsymbol{y}$
- We want to find the **optimum** $\widehat{\boldsymbol{\theta}}^*$ such that:

$$\widehat{\boldsymbol{\theta}}^* = \underset{\widehat{\boldsymbol{\theta}}}{\operatorname{argmin}}\, L(\widehat{\boldsymbol{\theta}})$$

- Considering the change in the loss function with respect to $\widehat{\boldsymbol{\theta}}$, the cost function is minimum when $\frac{dL(\widehat{\boldsymbol{\theta}})}{d\widehat{\boldsymbol{\theta}}} = 0$.
- The Least Squares cost function for linear regression is ***convex quadratic.*** Optimal $\widehat{\boldsymbol{\theta}}$ can be found by solving its **Normal Equation.**





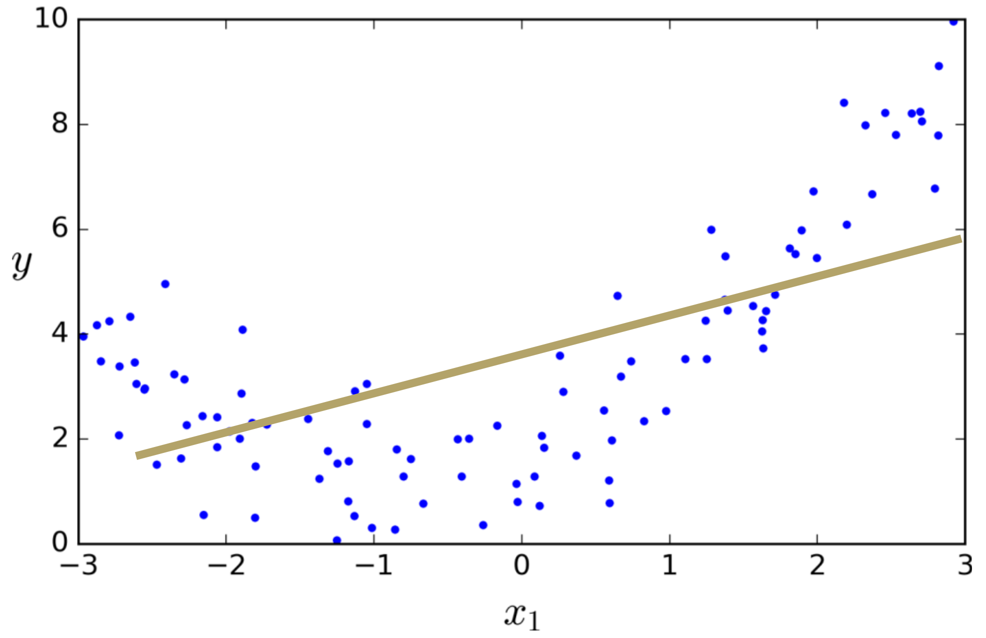


Overview

In this Lecture..

Nonlinear Regression
Introduction

- When the underlying relationship between input and output variables is **non-linear**, a linear regression model fails to fit the data.



Linear regression estimating **non-linear** relationship

Nonlinear Regression
Introduction

- Recall that we can define a kernel $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \phi(\boldsymbol{x}_i)^\top \phi(\boldsymbol{x}_j)$ as a *feature transformation* function to extend linear SVMs to non-linear classifier, e.g., SVMs with a polynomial kernel
- With the same intuition, linear regression can be extended to model **non-linear** relationships by performing **non-linear *feature transformation*** on the input $\boldsymbol{x}$ via a non-linear function, $\phi(\boldsymbol{x})$.
- A nonlinear regression model with a single non-linear function $\phi(\boldsymbol{x})$ takes the form:

$$y_i = \theta_0 + \theta_1 \phi(\boldsymbol{x}_i)$$

Nonlinear Regression

Introduction

- Generally, a more complex nonlinear regression model uses multiple non-linear functions $\phi(\boldsymbol{x})$. The model can be formed as a weighted sum of m nonlinear functions $\{\phi_1(\boldsymbol{x}), \phi_2(\boldsymbol{x}),\ldots, \phi_m(\boldsymbol{x})\}$:

$$y_i = \theta_0 + \theta_1\phi_1(\boldsymbol{x_i}) + \theta_2\phi_2(\boldsymbol{x_i}) + \cdots + \theta_m\phi_m(\boldsymbol{x}_i), \qquad i = 1, \ldots, N$$

where $\phi_1(\boldsymbol{x}),\ldots, \phi_m(\boldsymbol{x})$ are nonlinear functions or *feature transformations*, $\boldsymbol{\theta} = [\theta_0, \theta_1, \theta_2 \ldots \theta_m]^T$ is the associated weights vector

- In analogy to the linear regression, the generic nonlinear regression can be modeled as:

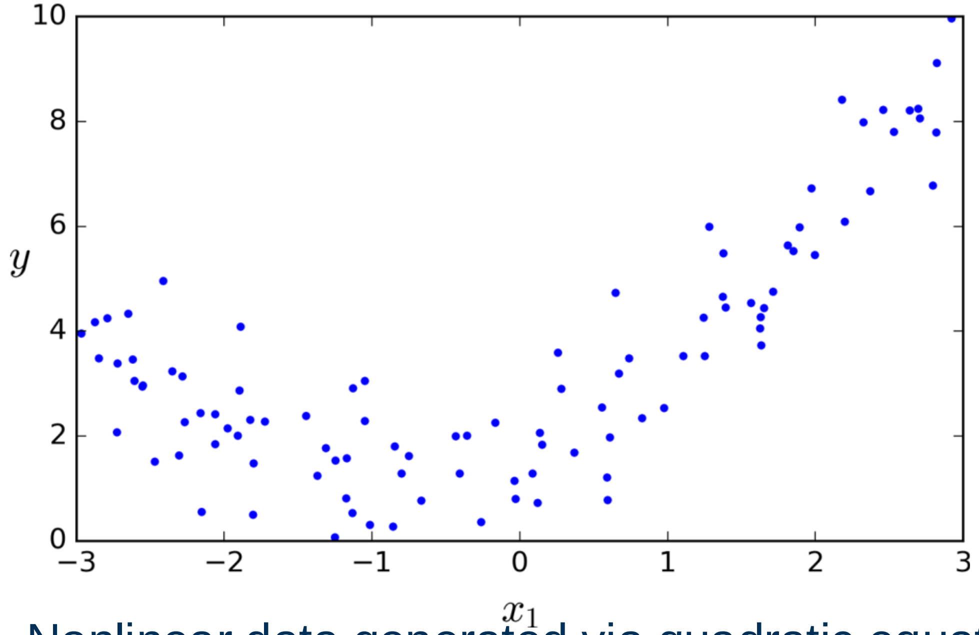
$$y_{\boldsymbol{i}} = \boldsymbol{\theta}^{\mathbf{T}}\boldsymbol{\phi}(\boldsymbol{x}_i)$$

where $\boldsymbol{\phi}(\boldsymbol{x}_i) = \begin{bmatrix} 1 \\ \phi_1(\boldsymbol{x_i}) \\ \vdots \\ \phi_m(\boldsymbol{x_i}) \end{bmatrix}, \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_m \end{bmatrix}$

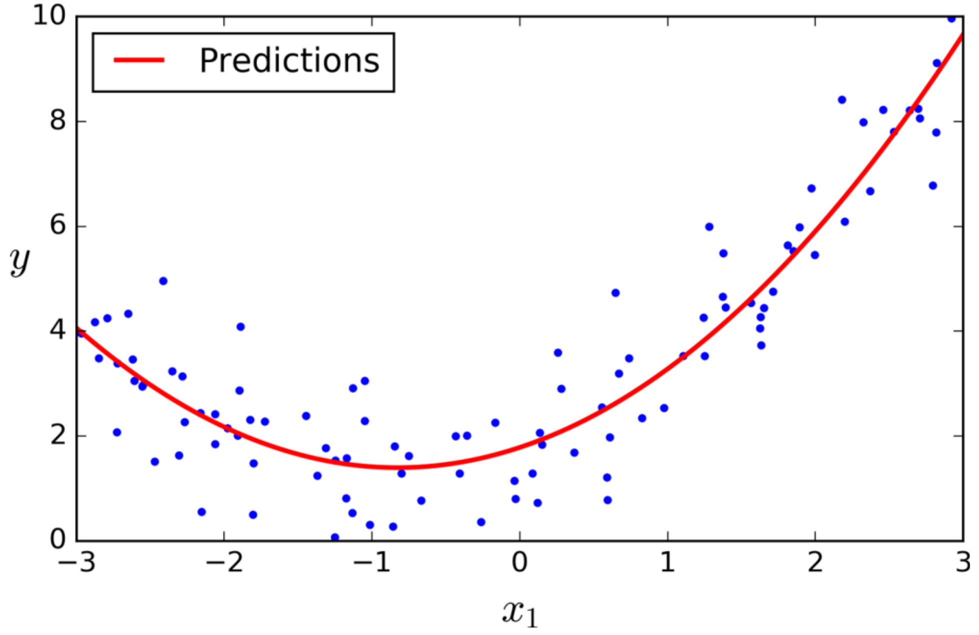
Nonlinear Regression

Polynomial Regression

- Assumes non-linear relationship between the output variable y and the input variable $\boldsymbol{x}$
- $\boldsymbol{\phi}(\boldsymbol{x}_i)$ consists of power functions, i.e., $\boldsymbol{\phi}(\boldsymbol{x}_i) = [1, \phi_1(\boldsymbol{x}_i), \dots, \phi_m(\boldsymbol{x}_i)]$, where $\phi_j(x_i) = (x_i)^j$



Nonlinear data generated via quadratic equation ($y = ax^2 + bx + c$) with random noise



Polynomial regression model prediction fitted to data

Polynomial Regression
High-degree Polynomial Regression

- For the case where samples have single feature, the non-linear relationship is modeled as the following form:

$$y_i = \hat{\theta}_0 + \hat{\theta}_1 x_i + \hat{\theta}_2 x_i^2 + \cdots + \hat{\theta}_m x_i^m + \varepsilon_i, \qquad i = 1, \dots, N$$

where $\boldsymbol{\phi}(\boldsymbol{x}_i) = [1, x_i, \dots, x_i^m]$is a vector of the features extended to m degrees. The relationship can be expressed as a system of linear equations :

$$\boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_N \end{bmatrix}, \boldsymbol{\phi}(\boldsymbol{x}) = \begin{bmatrix} \boldsymbol{\phi}(x_1) \\ \boldsymbol{\phi}(x_2) \\ \boldsymbol{\phi}(x_3) \\ \vdots \\ \boldsymbol{\phi}(x_N) \end{bmatrix} = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^m \\ 1 & x_2 & x_2^2 & \cdots & x_2^m \\ 1 & x_3 & x_3^2 & \cdots & x_3^m \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_N & x_N^2 & \cdots & x_N^m \end{bmatrix}, \hat{\boldsymbol{\theta}} = \begin{bmatrix} \hat{\theta}_0 \\ \hat{\theta}_1 \\ \hat{\theta}_2 \\ \vdots \\ \hat{\theta}_m \end{bmatrix}, \boldsymbol{\varepsilon} = \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \\ \vdots \\ \varepsilon_N \end{bmatrix},$$

where m is the polynomial's degree, $\boldsymbol{\varepsilon}$ is the residual error.

Using matrix notation, it can be expressed as

$$\boldsymbol{y} = \boldsymbol{\phi}(\boldsymbol{x})\hat{\boldsymbol{\theta}} + \boldsymbol{\varepsilon}$$

Polynomial Regression

High-degree Polynomial Regression

- What if we have multiple features? Say $P = 2$(no. of features) & $m = 2$(polynomial degree)

$$y_i = \hat{\theta}_0 + \hat{\theta}_1 x_{i,1} + \hat{\theta}_2 x_{i,2} + \hat{\theta}_3 x_{i,1}^2 + \hat{\theta}_4 x_{i,2}^2 + \hat{\theta}_5 x_{i,1} x_{i,2} + \varepsilon_i$$

$$\boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_N \end{bmatrix}, \boldsymbol{\phi}(\boldsymbol{X}) = \begin{bmatrix} \boldsymbol{\phi}(x_1) \\ \boldsymbol{\phi}(x_2) \\ \boldsymbol{\phi}(x_3) \\ \vdots \\ \boldsymbol{\phi}(x_N) \end{bmatrix} = \begin{bmatrix} 1 & x_{1,1} & x_{1,2} & x_{1,1}^2 & x_{1,2}^2 & x_{1,1}x_{1,2} \\ 1 & x_{2,1} & x_{2,2} & x_{2,1}^2 & x_{3,2}^2 & x_{2,1}x_{2,2} \\ 1 & x_{3,1} & x_{3,2} & x_{3,1}^2 & x_{3,2}^2 & x_{3,1}x_{3,2} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{N,1} & x_{N,2} & x_{N,1}^2 & x_{N,2}^2 & x_{N,1}x_{N,2} \end{bmatrix}, \boldsymbol{\theta} = \begin{bmatrix} \hat{\theta}_0 \\ \hat{\theta}_1 \\ \hat{\theta}_2 \\ \hat{\theta}_3 \\ \hat{\theta}_4 \\ \hat{\theta}_5 \end{bmatrix}, \boldsymbol{\varepsilon} = \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \\ \vdots \\ \varepsilon_N \end{bmatrix},$$

Polynomial Regression
Least Squares Cost Function

- Recall that for the linear regression *predictor* $\boldsymbol{y} = \boldsymbol{X}\widehat{\boldsymbol{\theta}} + \boldsymbol{\varepsilon}$, we assume that $\boldsymbol{\varepsilon}$ is i.i.d. distributed and drawn from a Gaussian (normal) distribution. We derive the least squares cost function via the conditional PDF of linear regression:

$$p(y_i|\boldsymbol{x}_i;\boldsymbol{\theta}) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{\left(y_{\boldsymbol{i}} - \boldsymbol{\theta}^T\boldsymbol{x}_i\right)^2}{2\sigma^2}\right)$$

- For the polynomial regression predictor $\boldsymbol{y} = \boldsymbol{\phi}(\boldsymbol{x})\widehat{\boldsymbol{\theta}} + \boldsymbol{\varepsilon}$, we make a similar assumption, and the conditional PDF is:

$$p(y_i|\boldsymbol{x}_i;\boldsymbol{\theta}) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{\left(y_{\boldsymbol{i}} - \boldsymbol{\theta}^T\boldsymbol{\phi}(\boldsymbol{x}_{\boldsymbol{i}})\right)^2}{2\sigma^2}\right)$$

- The Least Squares cost function can be derived as:

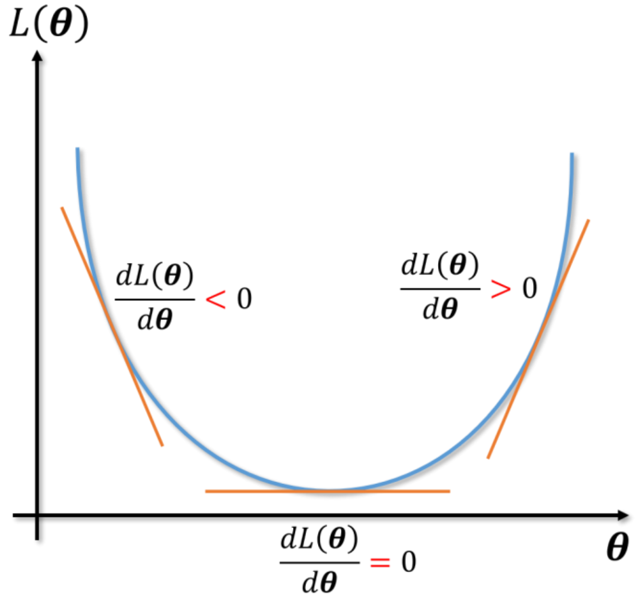
$$L(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2 = \frac{1}{N}\sum_{i=1}^{N}\left(\boldsymbol{\theta}^T\boldsymbol{\phi}(\boldsymbol{x}_i) - y_i\right)^2$$

Polynomial Regression
Finding Optimal Parameters

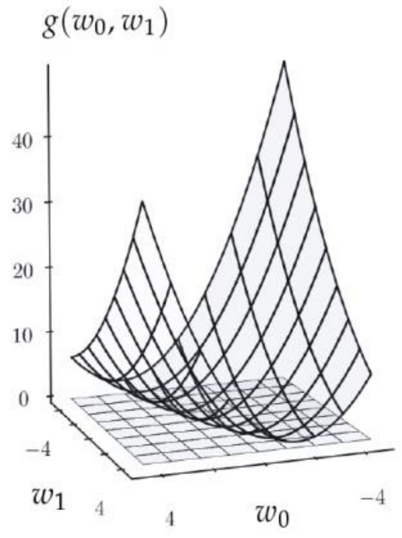
- Let $L(\boldsymbol{\theta})$ be the cost function that measures the difference between predictions $\hat{\boldsymbol{y}}$ and desired outputs $\boldsymbol{y}$
- We want to find the **optimum $\boldsymbol{\theta}^*$** such that:

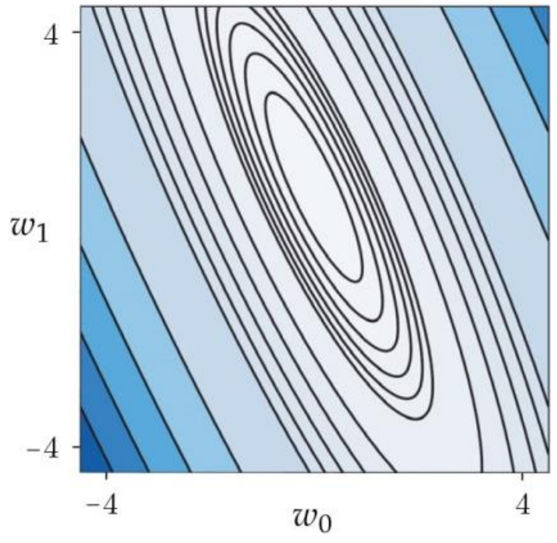
$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} L(\boldsymbol{\theta})$$

- Considering the change in the loss function with respect to $\boldsymbol{\theta}$, the cost function is minimum when $\frac{dL(\boldsymbol{\theta})}{d\boldsymbol{\theta}} = 0$.
- The Least Squares cost function for linear regression is ***convex quadratic.*** Optimal $\boldsymbol{\theta}$ can be found by solving its **Normal Equation.**



$L(\boldsymbol{\theta})$
$\frac{dL(\boldsymbol{\theta})}{d\boldsymbol{\theta}} < 0$
$\frac{dL(\boldsymbol{\theta})}{d\boldsymbol{\theta}} > 0$
$\frac{dL(\boldsymbol{\theta})}{d\boldsymbol{\theta}} = 0$
$\boldsymbol{\theta}$





Polynomial Regression
The Normal Equation

- The least squares for polynomial regression can be written in matrix form as:

$$\boldsymbol{L}(\boldsymbol{\theta}) = \frac{1}{N}\left(\boldsymbol{\phi}(\boldsymbol{X})\widehat{\boldsymbol{\theta}} - \boldsymbol{y}\right)^T\left(\boldsymbol{\phi}(\boldsymbol{X})\widehat{\boldsymbol{\theta}} - \boldsymbol{y}\right)$$

- when $\frac{\partial J_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} = 0,\ \boldsymbol{\theta} = \left(\boldsymbol{\phi}(\boldsymbol{X})^T\boldsymbol{\phi}(\boldsymbol{X})\right)^{-1}\boldsymbol{\phi}(\boldsymbol{X})^T\boldsymbol{y}$
- For the case where samples have single feature, i.e.,$\boldsymbol{\phi}(\boldsymbol{x}_i) = [1, x_{\boldsymbol{i}}, \dots, x_{\boldsymbol{i}}^m] \in \mathbb{R}^{m+1}$, the Normal Equation handles efficiently
- However, for general case where $\boldsymbol{x}_i \in \mathbb{R}^P$, $\boldsymbol{\phi}(\boldsymbol{x}_i) = \{\prod_{j=1}^P x_{ij}^{b_j} : \sum_{j=1}^P b_j \leq m\} \in \mathbb{R}^{\binom{P+m}{m}}$ where b_j is the power degree for $x_{ij}^{b_j}$
- e.g., $\boldsymbol{x}_i = [x_{i1}, x_{i2}]^T \in \mathbb{R}^2$, $\boldsymbol{\phi}(\boldsymbol{x}_i)$ for 3-degree polynomial regression is:

$$\boldsymbol{\phi}(\boldsymbol{x}_i) = \left[1, x_{i1}, x_{i2}, x_{i1}x_{i2}, x_{i1}^2, x_{i2}^2, x_{i1}^2x_{i2}, x_{i1}x_{i2}^2, x_{i1}^3, x_{i2}^3\right] \in \mathbb{R}^{\binom{5}{3}}$$

- This requires solving the inverse of $\boldsymbol{\phi}(\boldsymbol{X})^T\boldsymbol{\phi}(\boldsymbol{X})$ which is a $\binom{P+m}{m}$x$\binom{P+m}{m}$ matrix

Polynomial Regression

Computational Complexity of The Normal Equation

- Requires finding the inverse of $\boldsymbol{\phi}(\boldsymbol{X})^T\boldsymbol{\phi}(\boldsymbol{X})$ which is a $\binom{P+m}{m}$x $\binom{P+m}{m}$ matrix, where P is the feature dimension and m is the polynomial degree
- Advantage:
  - Handles large data set efficiently where $N \gg \binom{P+m}{m}$. Linear with regards to number of instances, $O(N)$
- Disadvantage:
  - Finding the inverse is about $O\left(\binom{P+m}{m}^{2.4}\right)$ to $O\left(\binom{P+m}{m}^{3}\right)$
  - Gets very slow when the number of features grows large. e.g., when P=50, m=3, $\boldsymbol{\phi}(\boldsymbol{X})^T\boldsymbol{\phi}(\boldsymbol{X}) \in \mathbb{R}^{23426\times 23426}$. Moreover, gets intractable to even store in the memory.
- Alternative solution: Training by *Gradient Descent* to find optimal $\boldsymbol{\theta}$, which is better with large number of features

Overview

In this Lecture..

High-degree Polynomial Regression

Training by Gradient Descent

- The Algorithm
- Gradient Descent Alternatives
- Optimization techniques

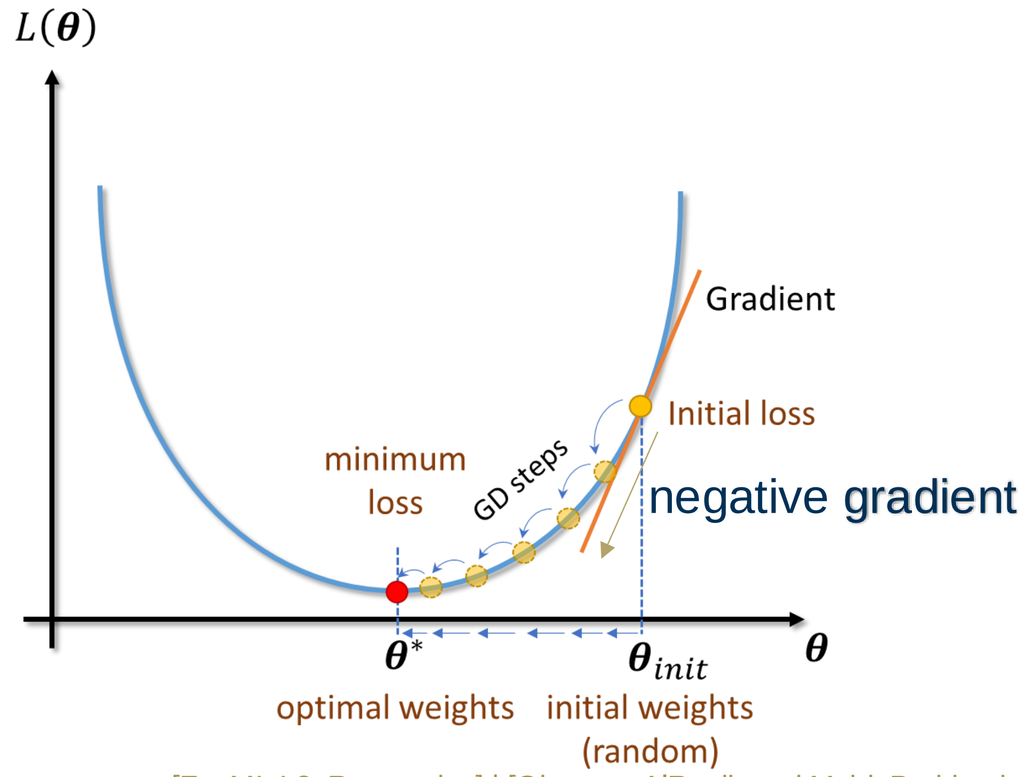
Regularized Regression Models

Performance Measures

Model Validation

Gradient Descent
Finding Optimal Parameters

- The Gradient Descent algorithm starts at an arbitrary $\boldsymbol{\theta}$ and iteratively follows the direction of the negative gradient to gradually find the minimum



The negative gradient (derivative) points to the direction of the greatest rate of decrease of the cost function. Its magnitude is the slope of the function in that direction

Gradient Descent
Gradients of Least Squares Loss Function

- The least squares loss function:

$$MSE(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2 = \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{x}_i^T\boldsymbol{\theta} - y_i)^2$$

- Partial derivative of the loss function:

$$\frac{\partial}{\partial\theta_p} MSE(\boldsymbol{\theta}) = \frac{2}{N}\sum_{i=1}^{N}(\boldsymbol{x}_i^T\boldsymbol{\theta} - y_i)\, x_{ip}$$

- Gradient vector of the loss function:

$$\nabla_\theta MSE(\boldsymbol{\theta}) = \begin{bmatrix} \frac{\partial}{\partial\theta_0} MSE(\boldsymbol{\theta}) \\ \frac{\partial}{\partial\theta_1} MSE(\boldsymbol{\theta}) \\ \vdots \\ \frac{\partial}{\partial\theta_P} MSE(\boldsymbol{\theta}) \end{bmatrix} = \frac{2}{N}\boldsymbol{X}^T(\boldsymbol{X\theta} - \boldsymbol{y})$$

- Gradient Descent step:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha\nabla_\theta MSE(\boldsymbol{\theta})$$

where α is the learning rate,
(t) and (t+1) are current and next iterations, respectively.

Gradient Descent

Gradients of Least Squares Loss Function

The gradient descent update rule:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha \nabla_\theta MSE(\boldsymbol{\theta})$$

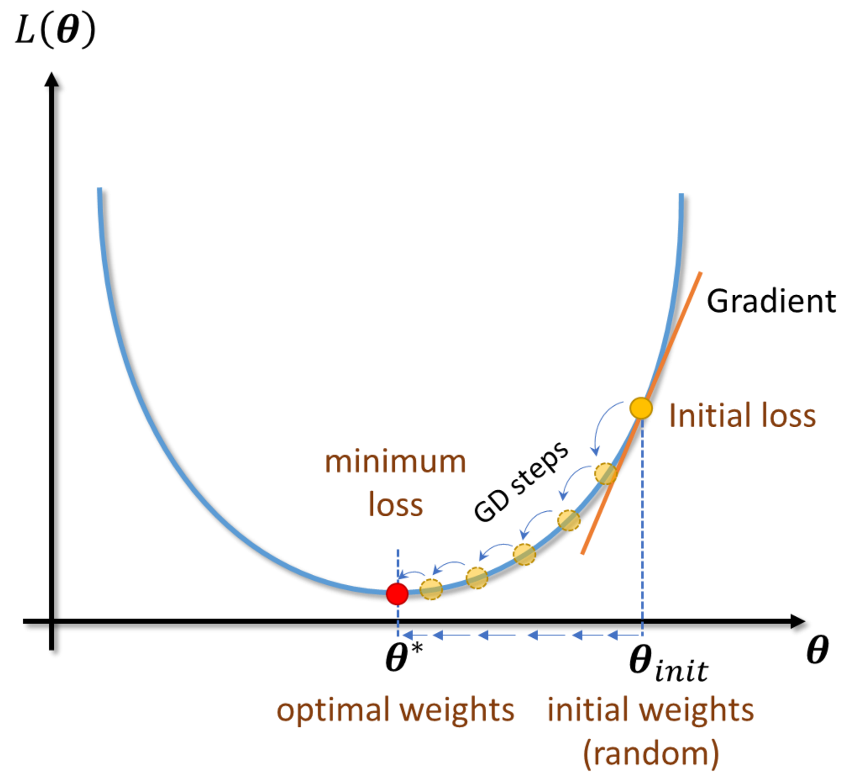
where $\nabla_\theta MSE(\boldsymbol{\theta}) = \frac{2}{N} \boldsymbol{X}^T (\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y})$

where α is the learning rate,
(t) and (t+1) are current and next iterations, respectively.

For polynomial regression, the gradient descent will take the similar form:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha \nabla_\theta MSE(\boldsymbol{\theta})$$

where $\nabla_\theta MSE(\boldsymbol{\theta}) = \frac{2}{N} \boldsymbol{\phi}(\boldsymbol{X})^T (\boldsymbol{\phi}(\boldsymbol{X})\boldsymbol{\theta} - \boldsymbol{y})$,
$\boldsymbol{\phi}(\boldsymbol{X}) \in \mathbb{R}^{N \times \binom{P+m}{m}}$



Gradient Descent
Finding Optimal Parameters

Calculating $\nabla_{\theta} MSE(\boldsymbol{\theta}) = \frac{2}{N}\boldsymbol{X}^T(\boldsymbol{X\theta} - \boldsymbol{y})$ needs to compute the gradient over the entire training set at every step, which makes it very slow when the training set is large.

There are other alternatives for implementing the gradient descent algorithm without computing the derivative over the entire training set:

- ***Batch** Gradient Descent:*

  Calculate gradients of cost function over all instances at each step

- ***Stochastic** Gradient Descent*

  Calculate gradients of cost function over a single random instance at each step

- ***Mini-batch** Gradient Descent*

  Calculate gradients of cost function over every small batch of instances

Training is performed in iterations in which the training set is processed to calculate gradual adjustments to coefficients in small steps until convergence.

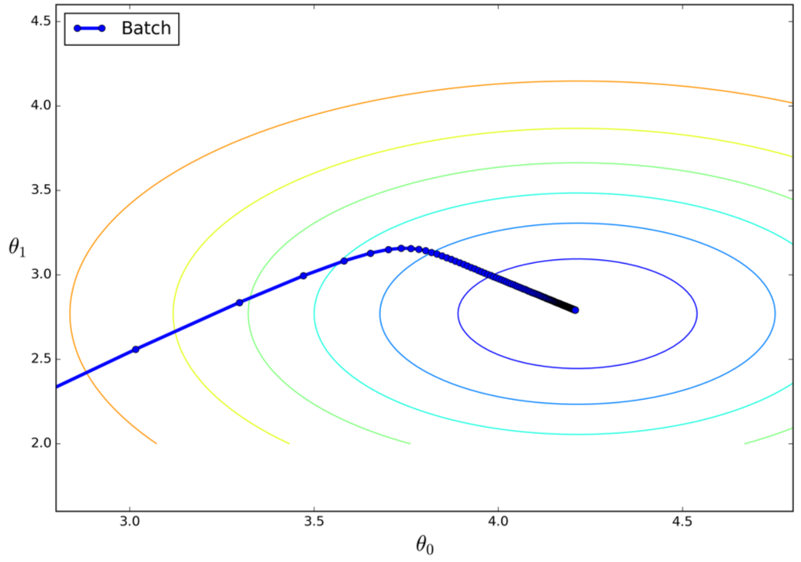
Gradient Descent

Batch Gradient Descent

```
Randomly initialize coefficient vector $\theta$
Repeat until convergence
{
    Calculate gradient $\nabla_\theta MSE(\theta)$ over the entire
    dataset: $\nabla_\theta MSE(\boldsymbol{\theta}) = \frac{2}{N}\boldsymbol{X}^T(\boldsymbol{X\theta} - \boldsymbol{y})$
    Update $\theta^{(t+1)} = \theta^{(t)} - \alpha\nabla_\theta MSE(\theta)$

     theta = theta -(2/N)*learning_rate*(
np.matmul(X.T, (np.matmul(X,theta)-y)) )
}
```

Convergence path of Batch GD



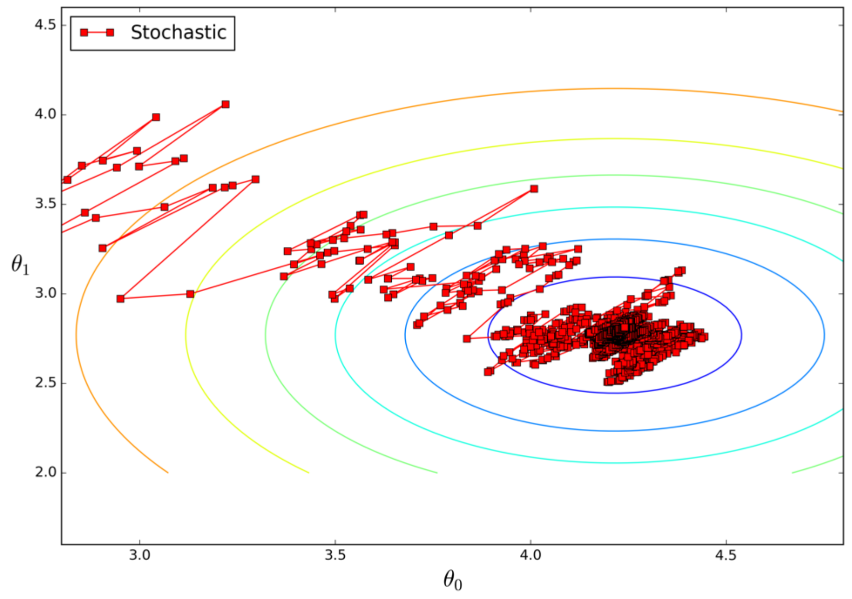
- Convergence takes place when $\|\nabla_\theta MSE(\boldsymbol{\theta})\| < \epsilon$, where ϵ (tolerance) takes a tiny value. It indicates that coefficients will almost not update with further steps
- Smoother convergence

Gradient Descent

Stochastic Gradient Descent

```
Initialize coefficient vector $\theta$
Repeat until convergence
{
  **For N iterations:**
    {Randomly pick *i-th sample* $\boldsymbol{x}_i$
    Calculate $\nabla_\theta MSE(\theta, \boldsymbol{x}_i) = 2(\boldsymbol{x}_i^T\boldsymbol{\theta} - y_i)\boldsymbol{x}_i$
    Update $\theta^{(t+1)} = \theta^{(t)} - \alpha\nabla_\theta MSE(\theta)$
    theta = theta-2*learning_rate*(np.matmul(X_i.T,
(np.matmul(X_i,theta)-y_i)))
    }
}
```

Convergence path of Stochastic GD



- Much faster because gradient is calculated based on a single sample
- Convergence path is much more stochastic
- Final coefficient values are good, but may not be optimal
- When cost function is non-convex, bouncing might help algorithm escape local minima

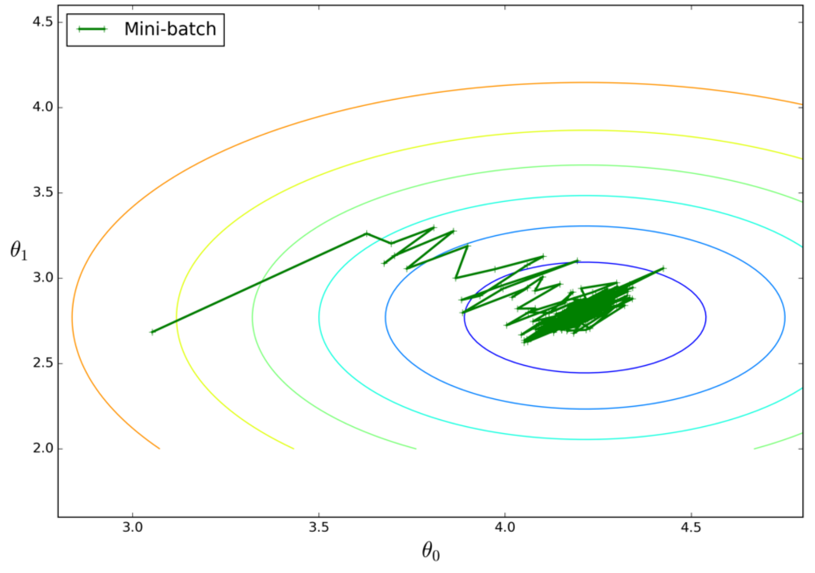
Gradient Descent

Mini-batch Gradient Descent

```
Initialize coefficient vector $\theta$
Repeat until convergence
{
  For N/b iterations:
    {Sample a subset $\boldsymbol{X}_S$ of size $b$ instances
    at random
    Calculate $\nabla_\theta MSE(\theta, \boldsymbol{X}_S)$ over all
    samples in $\boldsymbol{X}_S$:  $\nabla_\theta MSE(\theta, \boldsymbol{X}_S) = \frac{2}{b}\boldsymbol{X}_S{}^T(\boldsymbol{X}_S\boldsymbol{\theta} - \boldsymbol{y})$
    Update $\theta^{(t+1)} = \theta^{(t)} - \alpha\nabla_\theta MSE(\theta)$
  }
}
```

```
theta = theta -
(2/b)*learning_rate*(np.matmul(X[i:i+b].T,
(np.matmul(X[i:i+b],theta)-y_i)))
```

Convergence path of mini-batch GD



- Trade-off between computation complexity and finding optimal coefficients
- Less erratic progress in coefficient space
- May be harder to escape local minima
- Allows for a performance boost from parallel hardware

Gradient Descent
Epoch vs Iteration

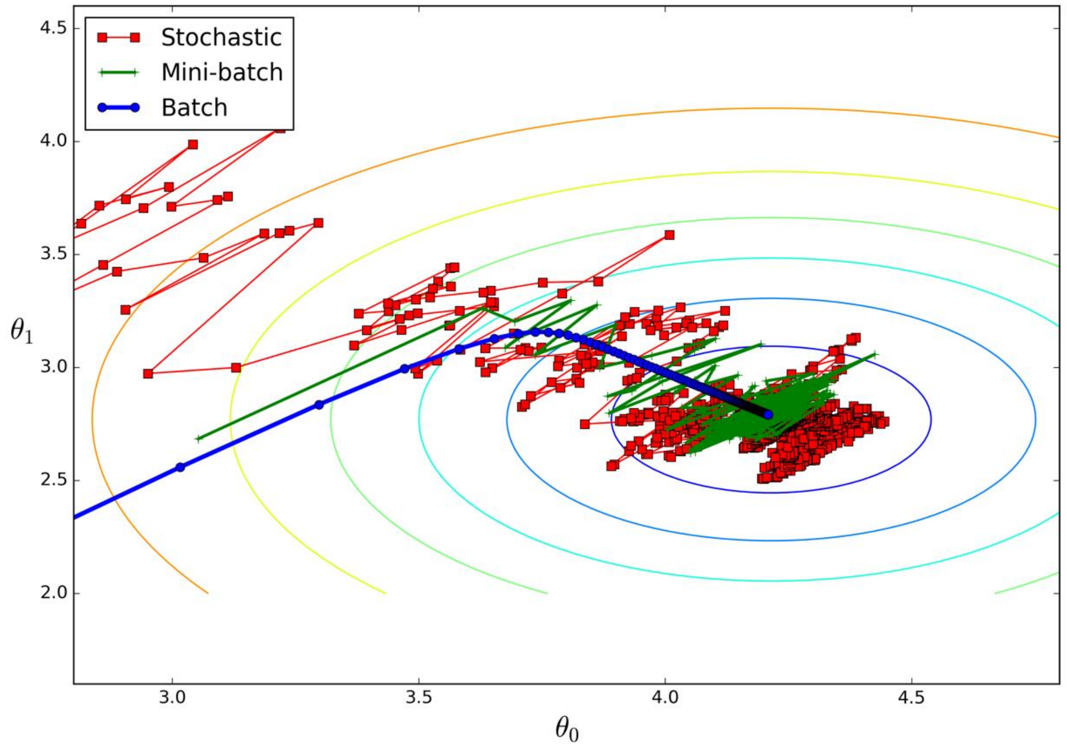
- **Epoch**: a sequence of repetitions which completes the processing of the entire dataset
- **Iteration**: processing samples to calculate a single gradient descent step and update coefficients

	Batch GD	Stochastic GD	Mini-batch GD
In one iteration:	The entire dataset is processed to calculate a GD step in a single iteration.	a single data sample is processed at random to calculate GD step and update coefficient.	A mini-patch subset of dataset is processed in one iteration (dataset is divided into b batches).
An epoch completes:	in one iteration.	in N iterations (when all data samples are processed)	when all batches are processed an epoch is completed (N/b iterations)

Gradient Descent

Convergence

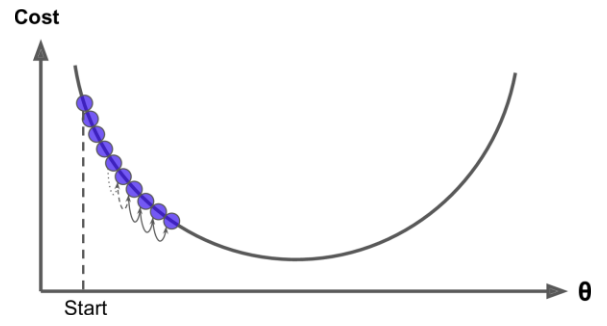
- All GD methods end up near the minimum
- Batch GD's path stops at the minimum
- Both Stochastic GD and Mini-batch GD continue to walk around.
- However, Batch GD takes a lot of time to take each step
- Stochastic GD and Mini-batch GD would also reach the minimum if a good learning schedule is used.

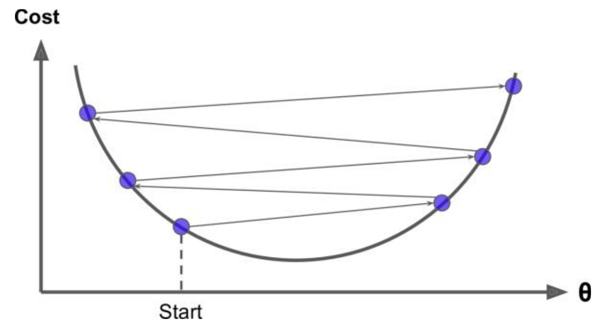


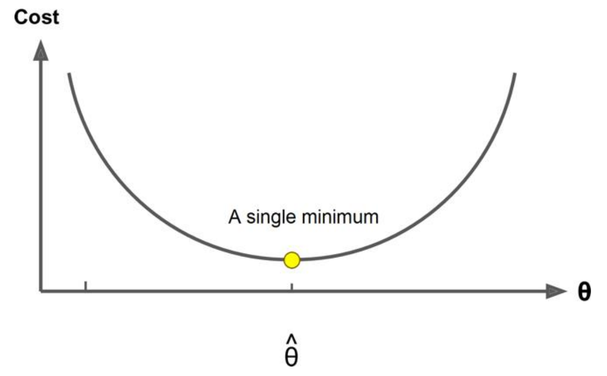
Gradient Descent

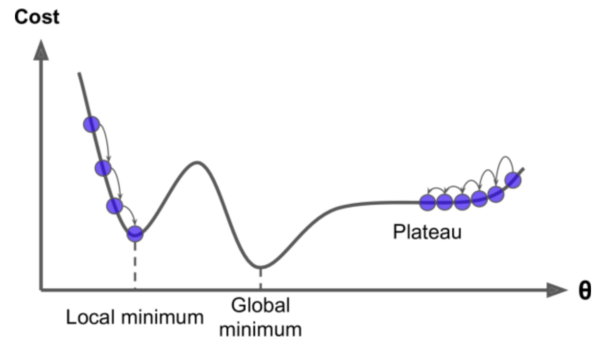
Pitfalls

- **Step size** (learning rate) affects convergence speed and accuracy:
  - Too small → slow convergence
  - Too large → oscillates around minimum and does not converge precisely
- **Local Minima** Problem
  - MSE is a *convex* function and, therefore, has only one minimum.
  - *Non-convex* functions have more than one minima, and, depending on initialization, Gradient Descent may miss the global minimum and converge onto one of the local minima.







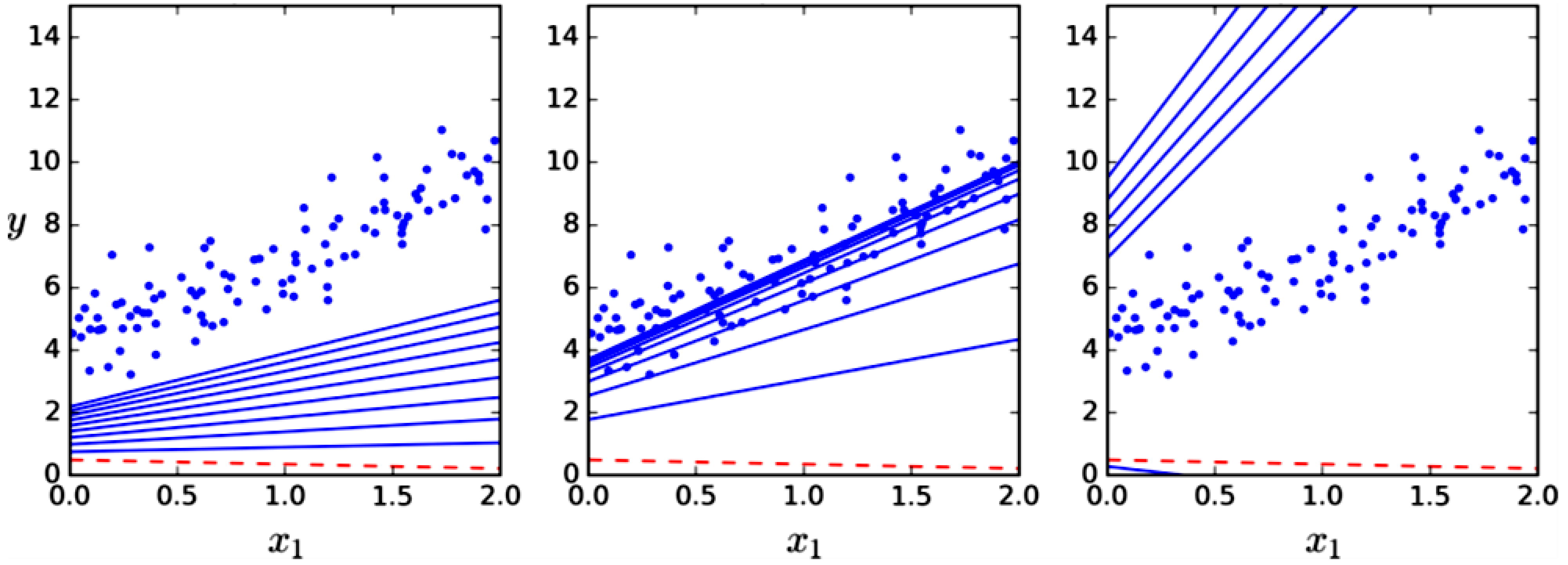


Gradient Descent

Batch Gradient Descent vs Learning Rates

- First 10 iterations of Batch Gradient Descent with various learning rates

$\alpha = 0.02$, converges too slow $\alpha = 0.1$, converges to the optimal $\alpha = 0.5$, diverges as stepping over the optimal

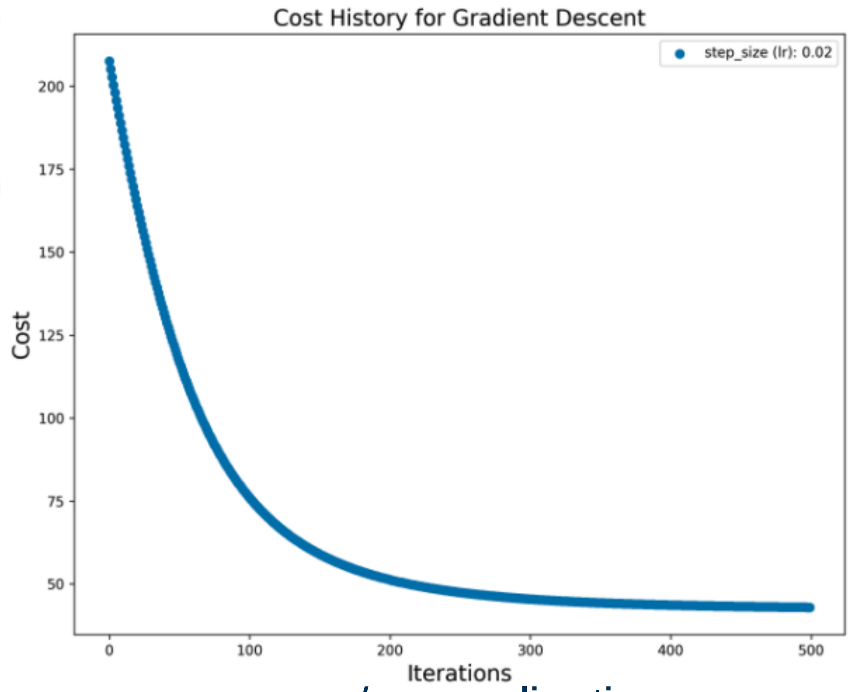


Red dotted line indicates initial model, each blue line is a model after every consecutive iteration (batch)

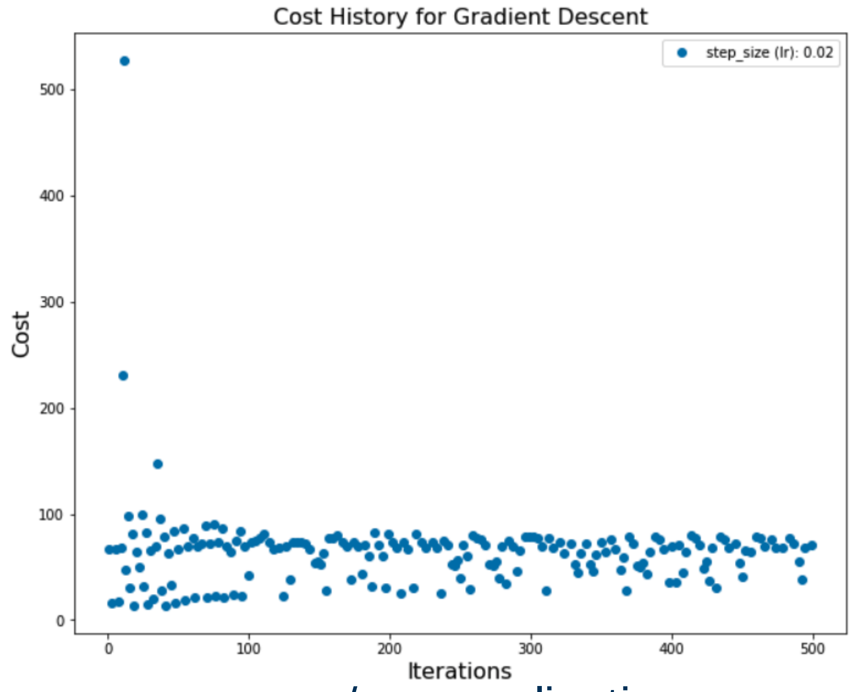
Gradient Descent

Feature Normalization

- To ensure that the gradient descent moves the cost smoothly towards the minima, the weights should be updated at the same rate by **normalizing the features** before training the model.
- One common normalization method is min-max normalization as $\boldsymbol{x}_{norm} =$
  $\boldsymbol{x} - min(\boldsymbol{x})$



w/ normalization



w/o normalization

Gradient Descent
Comparisons

N is the number of training instances and P is the number of features

Algorithm	Performance with Large N	Memory Space Requirements	Performance with Large P	Normalization required?
Normal Equation	Fast	High	Slow	No
Batch GD	Slow	High	Fast	Yes
Stochastic GD	Fast	Minimum	Fast	Yes
Mini-batch GD	Fast	Relative to batch size	Fast	Yes

Gradient Descent

Steepest Descent

$$f(a)+\frac{f'(a)}{1!}(x-a)+\frac{f''(a)}{2!}(x-a)^2+\frac{f'''(a)}{3!}(x-a)^3+\cdots,$$

- Taylor series approximation can be helpful when minimizing a function $L(\boldsymbol{\theta})$, which we do not have much knowledge about. Provided that the norm $\|\Delta\boldsymbol{\theta}\|_2$ is small, *i.e.*, $\boldsymbol{\theta}+\Delta\boldsymbol{\theta}$ is very close to $\boldsymbol{\theta}$, we can approximate the function $L(\boldsymbol{\theta}+\Delta\boldsymbol{\theta})$ by its first derivatives:

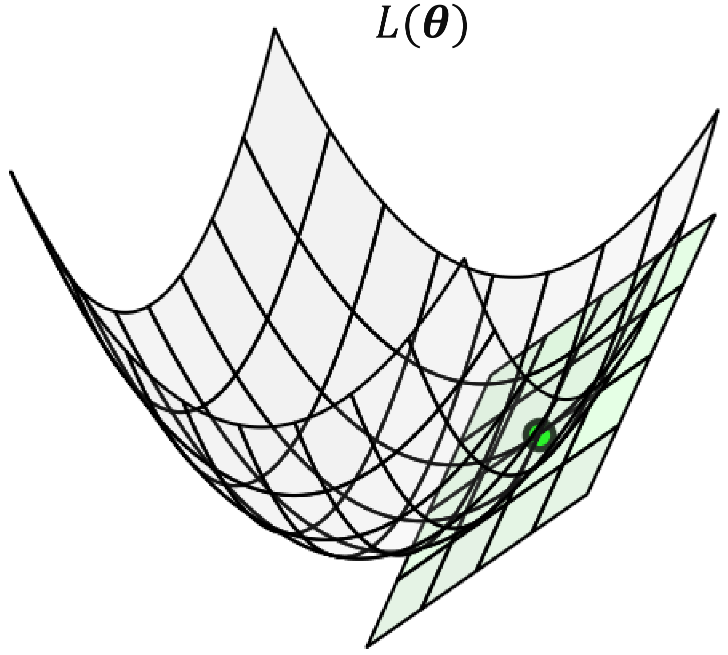
$$L(\boldsymbol{\theta}+\Delta\boldsymbol{\theta}) \approx L(\boldsymbol{\theta})+\Delta\boldsymbol{\theta}^T\nabla_\theta L(\theta)$$

where $\nabla_\theta L(\boldsymbol{\theta})$ is the gradient

- **Gradient Descent** uses the **first order** approximation**.** We assume that the function L around $\boldsymbol{\theta}$ can be approximated by a **linear hyper-plane** $L(\boldsymbol{\theta})+\Delta\boldsymbol{\theta}^T\nabla_\theta L(\theta)$

- For steepest descent we simply set $\Delta\boldsymbol{\theta}=-\boldsymbol{\alpha}\nabla_\theta L(\boldsymbol{\theta})$, which guarantees that $\boxed{L(\boldsymbol{\theta}+\Delta\boldsymbol{\theta}) \text{ decreases}}$ for $\boldsymbol{\alpha}>\mathbf{0}$:

$$L\big(\boldsymbol{\theta}-\boldsymbol{\alpha}\nabla_\theta L(\boldsymbol{\theta})\big)\approx L(\boldsymbol{\theta})-\underbrace{\boldsymbol{\alpha}\nabla_\theta L(\boldsymbol{\theta})^T\nabla_\theta L(\boldsymbol{\theta})}_{\text{positive}} < L(\boldsymbol{\theta})$$



Hyper-plane (green): $L(\boldsymbol{\theta})+\Delta\boldsymbol{\theta}^T\nabla_\theta L(\theta)$

Gradient descent:
$$\boldsymbol{\theta}^{t+1}=\boldsymbol{\theta}^t-\boldsymbol{\alpha}\nabla_\theta L(\boldsymbol{\theta}^t)$$

Appendix A: Notations

- x_i: a single feature
- $\boldsymbol{x}_i$: feature vector (a data sample)
- $\boldsymbol{x}_{:,i}$: feature vector of all data samples
- $\boldsymbol{X}$: matrix of feature vectors (dataset)
- N: number of data samples
- m: degree of polynomial
- P: number of features in a feature vector
- θ_i: a single model coefficient (parameter)
- $\boldsymbol{\theta}$: coefficient vector
- ε: error margin
- α: learning rate
- γ: bias factor
- Bold letter/symbol: vector
- Bold capital letters/symbol: matrix