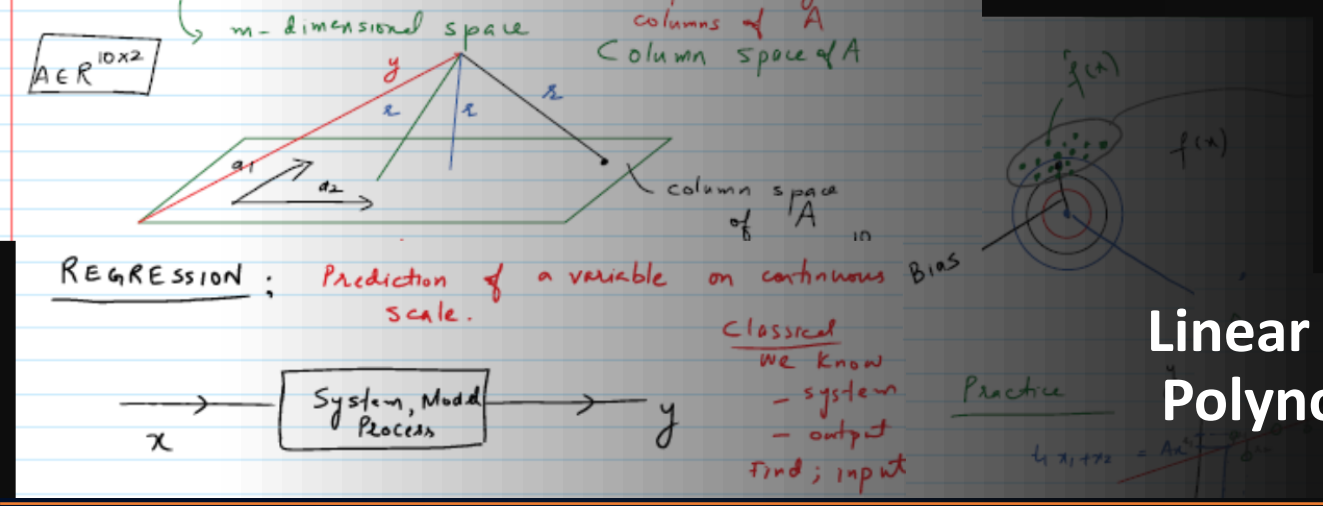


Machine Learning
EE514 – CS535

Linear Regression: Formulation, Solutions, Polynomial Regression, Gradient Descent and Regularization

Zubair Khalid

School of Science and Engineering
Lahore University of Management Sciences

https://www.zubairkhalid.org/ee514_2021.html

LUMS
A Not-for-Profit University

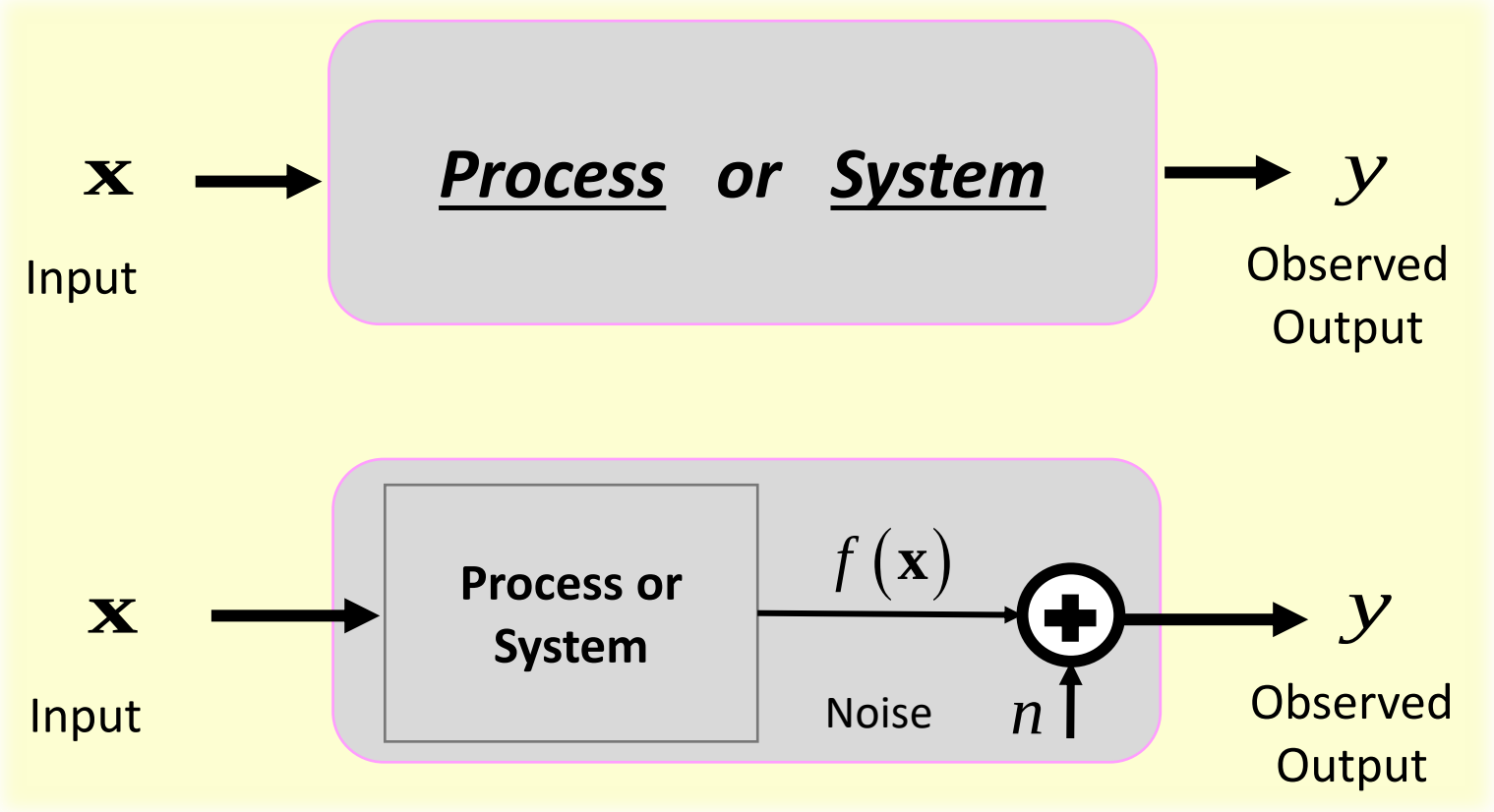
Outline

- Regression Set-up
- Linear Regression
- Polynomial Regression
- Underfitting/Overfitting
- Regularization
- Gradient Descent Algorithm

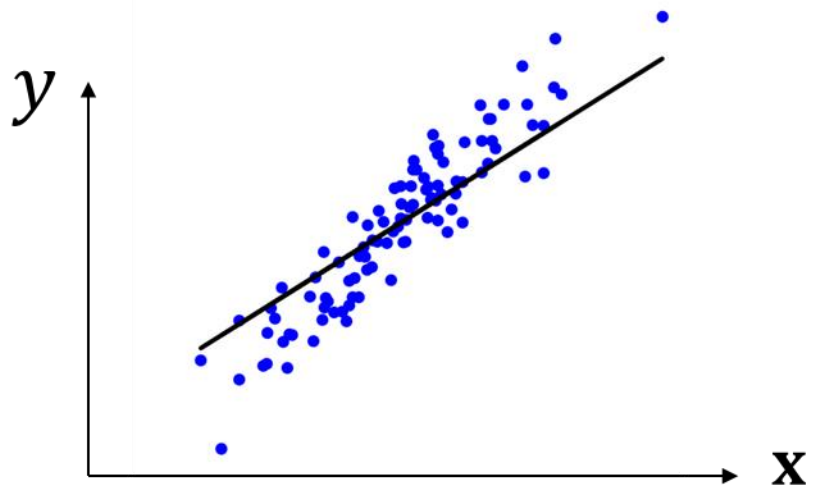
Regression

Regression: Quantitative Prediction on a continuous scale

- *Given a data sample, predict a numerical value*



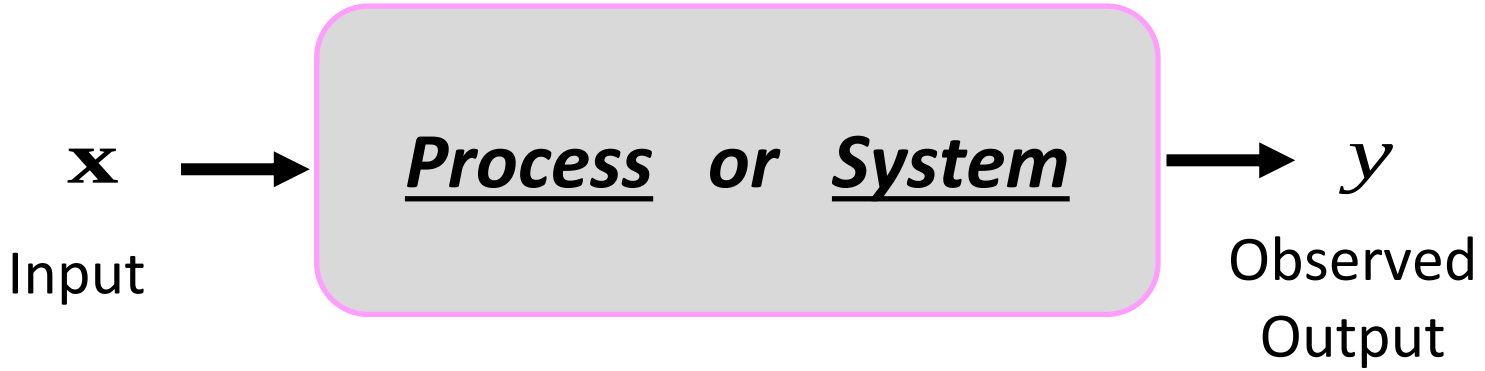
Example: Linear relationship



Here, PROCESS or SYSTEM refers to any underlying physical or logical phenomenon which maps our input data to our observed and noisy output data.

Regression

Overview:



One variable regression: y is a scalar

Multi-variable regression: $\underline{\mathbf{y}}$ is a vector

Single feature regression: $\mathbf{x}$ is a scalar

Multiple feature regression: $\underline{\mathbf{x}}$ is a vector

We will cover

Regression

Examples:

Single Feature:

- Predict score in the course given the number of hours of effort per week.
- Establish the relationship between the monthly e-commerce sales and the advertising costs.

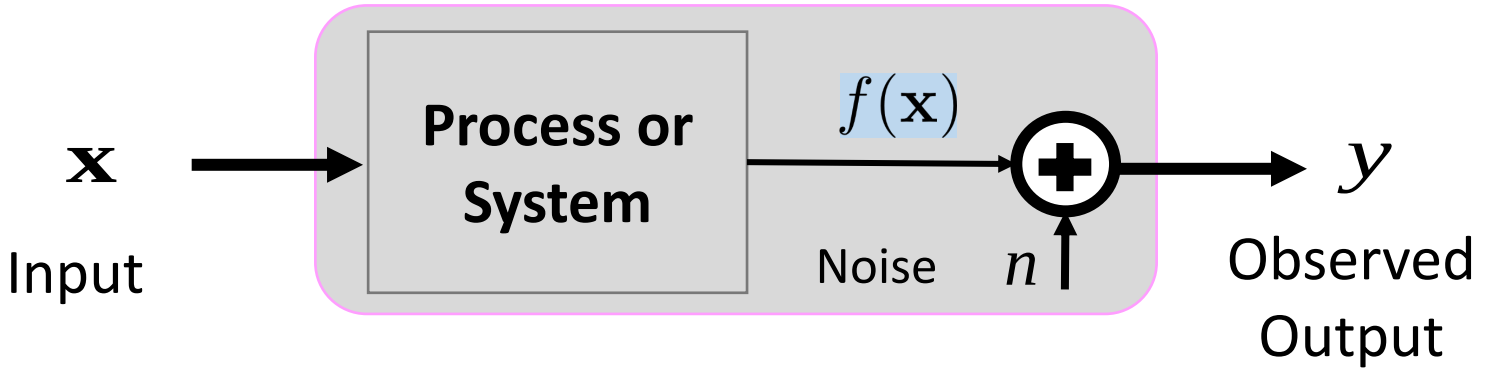
Multiple Feature:

- Studying operational efficiency of machine given sensors (temperature, vibration) data.
- Predicting remaining useful life (RUL) of the battery from charging and discharging information.
- Estimate sales volume given population demographics, GDP indicators, climate data, etc.
- Predict crop yield using remote sensing (satellite images, gravity information).
- Dynamic Pricing or Surge Pricing by ride sharing applications (Uber).
- Rate the condition (fatigue or distraction) of the driver given the video.
- Rate the quality of driving given the data from sensors installed on car or driving patterns.

Regression

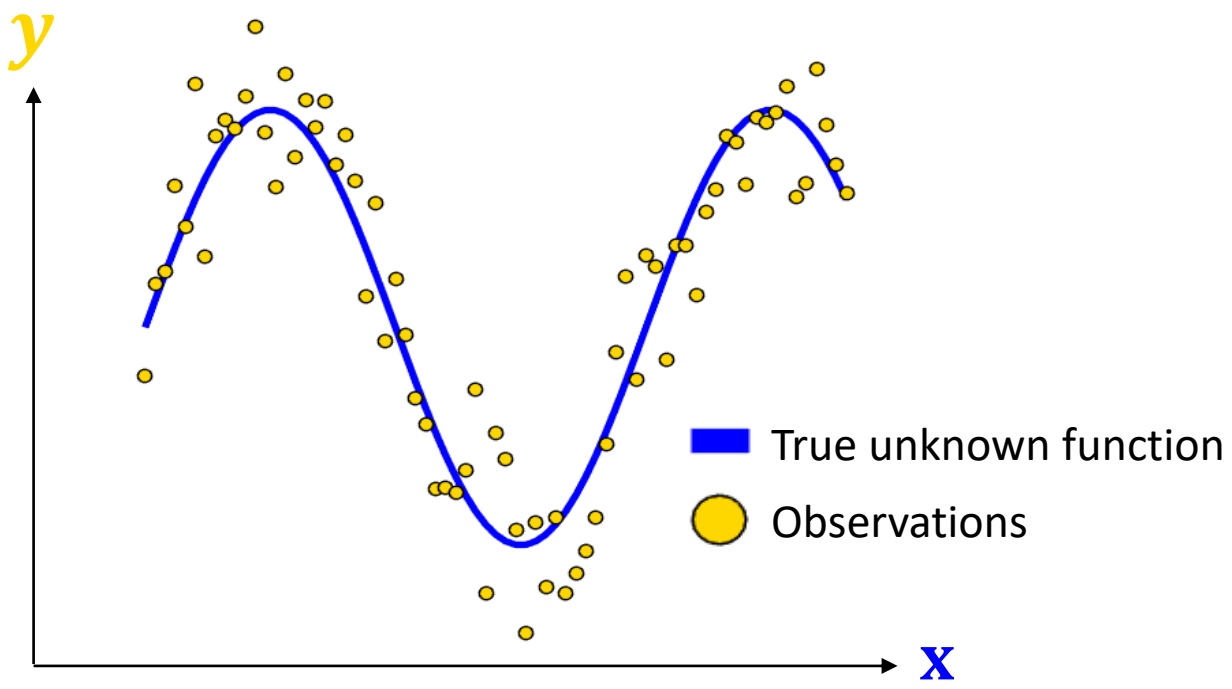
Model Formulation and Setup:

True Model:
We assume there is an inherent but **unknown** relationship between input and output.



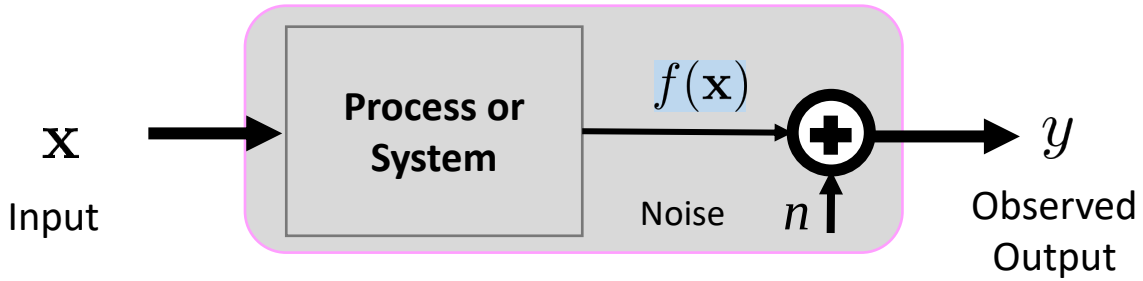
$$y = f(\mathbf{x}) + n$$

Goal:
Given noisy observations, we need to estimate **the unknown functional relationship** as accurately as possible.



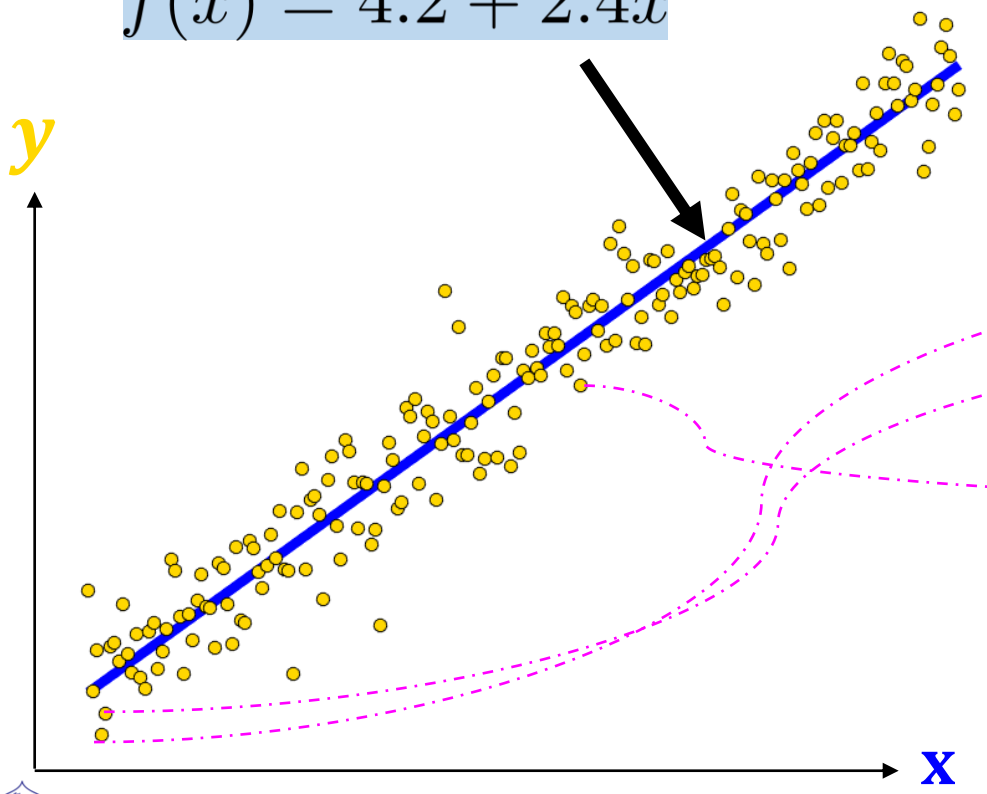
Regression

Model Formulation and Setup:



- *Single Feature Regression, Example:*

$$f(x) = 4.2 + 2.4x$$



Training Data

- First Data Sample: $\{\mathbf{x}(1), y(1)\}$
- Second Data Sample: $\{\mathbf{x}(2), y(2)\}$
- ...
- n-th Data Sample: $\{\mathbf{x}(n), y(n)\}$

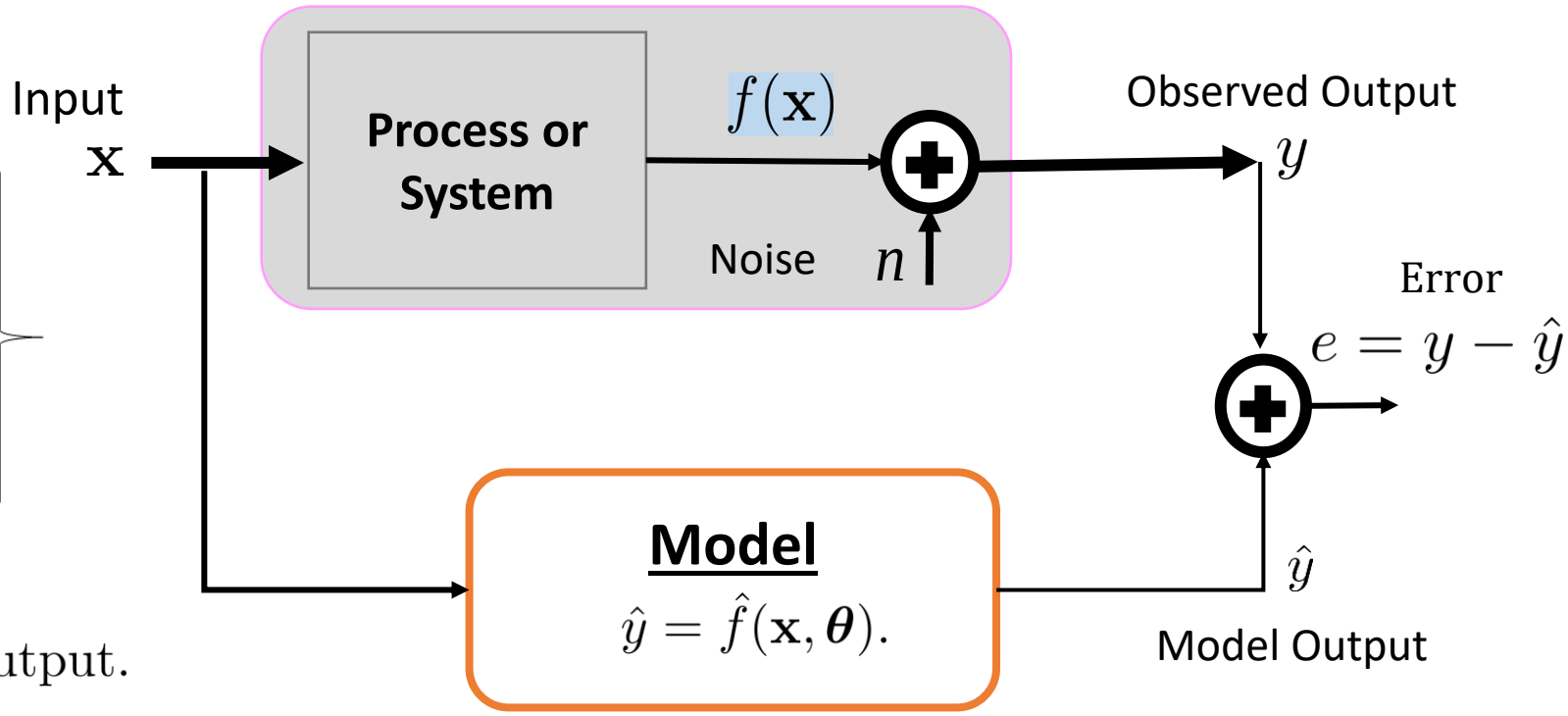
$$D = \{(\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2), \ldots, (\mathbf{x_n}, y_n)\} \subseteq \mathcal{X}^d \times \mathcal{Y}$$

Regression

Model Formulation and Setup:

We have:

- First Data Sample: $\{\mathbf{x}(1), y(1)\}$
- Second Data Sample: $\{\mathbf{x}(2), y(2)\}$
- ...
- n-th Data Sample: $\{\mathbf{x}(n), y(n)\}$



- For some input $\mathbf{x}$, $\hat{y}$ is our model output.
- Assume that our model is $\hat{f}(\mathbf{x}, \boldsymbol{\theta})$, characterized by the parameter(s) $\boldsymbol{\theta}$.
- Model $f(\mathbf{x}, \boldsymbol{\theta})$ has
  - A structure (e.g., linear, polynomial, inverse).
  - Paramaters in the vector $\boldsymbol{\theta} = [\theta_1, \theta_2, \ldots, \theta_M]$.
- Our model error is $e = y - \hat{y}$.

Linear Regression

Overview:

- Second learning algorithm of the course

- Scalar output is a linear function of the inputs

- Different from KNN: Linear regression adopts a modular approach which we will use most of the times in the course.
  - Select a model
  - Defining a loss function
  - Formulate an optimization problem to find the model parameters such that a loss function is minimized.
  - Employ different techniques to solve optimization problem or minimize loss function.

Linear Regression

Model:

We have $D = \{(\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2), \ldots, (\mathbf{x_n}, y_n)\} \subseteq \mathcal{X}^d \times \mathcal{Y}$

Model is a linear function of the features, that is,

$$\hat{f}(\mathbf{x}, \boldsymbol{\theta}) = \theta_0 + \sum_{i=1}^{d} \theta_i x_i \quad = \theta_0 + \boldsymbol{\theta}^T \mathbf{x}$$

- Linear structure.
- Model Paramaters: θ_0 and $\boldsymbol{\theta} = [\theta_1, \theta_2, \ldots, \theta_d]$.
  - θ_0 is bias or intercept.
  - $\boldsymbol{\theta} = [\theta_1, \theta_2, \ldots, \theta_d]$ represents the weights or slope.
  - θ_i quantifies the contribution of i-th feature x_i.

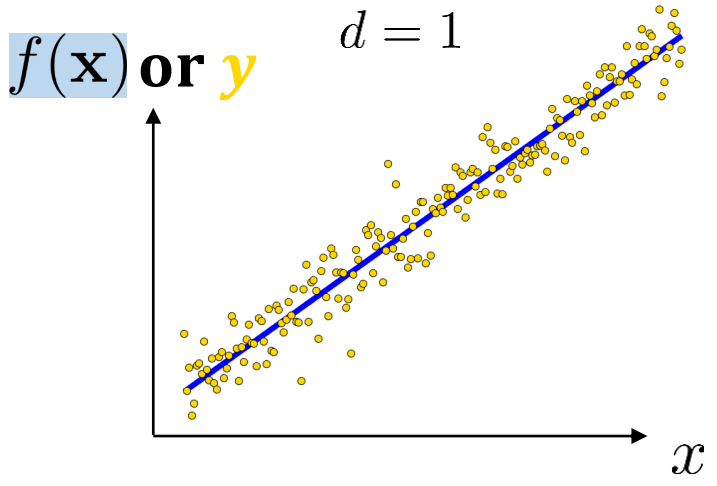
Linear Regression

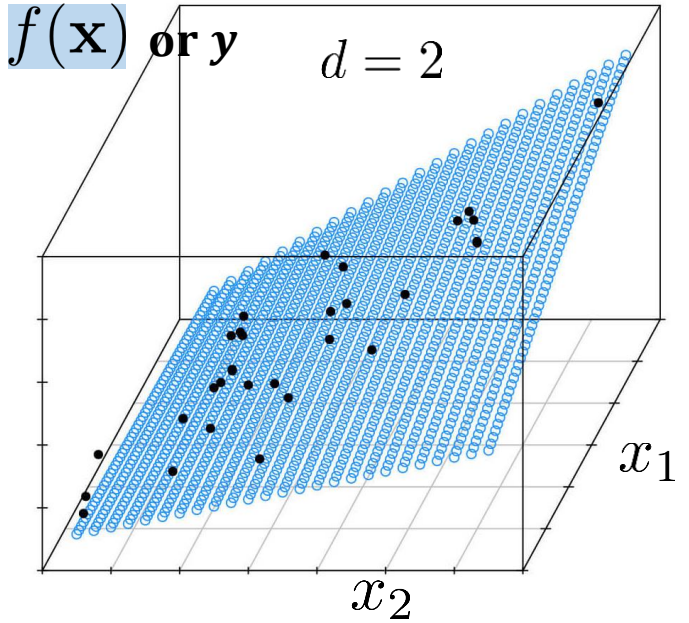
Model:

What is Linear?

Interpretation:

- $d = 1$ $\quad \hat{f}(\mathbf{x}, \boldsymbol{\theta}) = \theta_0 + \theta_1 x$ $\quad$ Line.
- $d = 2$ $\quad \hat{f}(\mathbf{x}, \boldsymbol{\theta}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$ $\quad$ Plane.
- d $\quad \hat{f}(\mathbf{x}, \boldsymbol{\theta}) = \theta_0 + \boldsymbol{\theta}^T \mathbf{x}$ $\quad$ Hyper-plane in $\mathbf{R}^{d+1}$





For different θ_0 and $\boldsymbol{\theta}$, we have different hyper-planes.

How do we find the 'best' line?

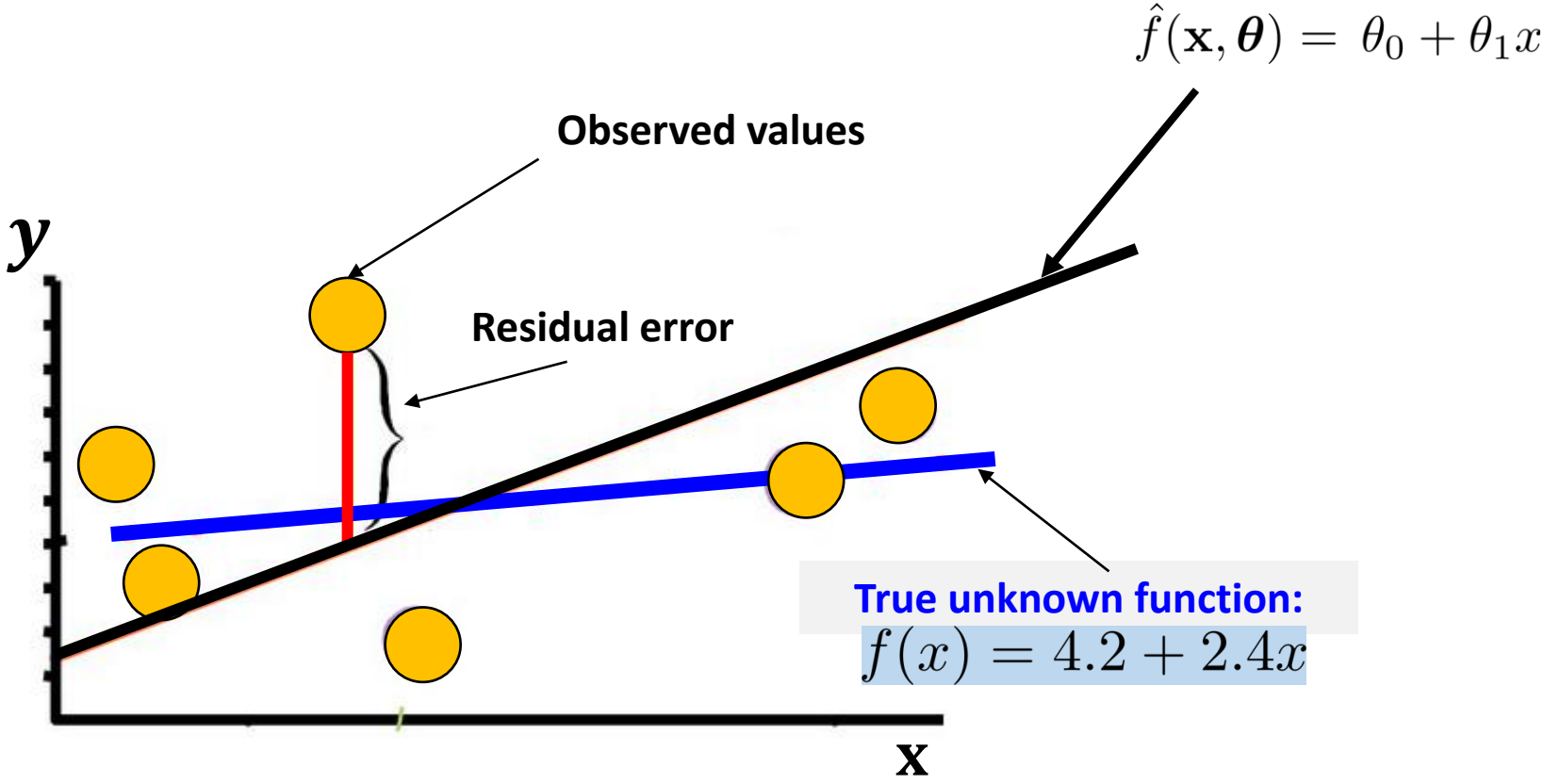
What do we mean by 'best'?

Linear Regression

Define Loss Function:

- *Loss function should be a function of model parameters.*

- For input $\mathbf{x}$, our model error is $e = y - \hat{y} = y - \hat{f}(\mathbf{x}, \boldsymbol{\theta}) = y - \theta_0 - \boldsymbol{\theta}^T \mathbf{x}$.
- e is also termed as residual error as it is the difference between observed value and predicted value.
- $d = 1$



Linear Regression

Define Loss Function:

- For $D = \{(\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2), \ldots, (\mathbf{x_n}, y_n)\} \subseteq \mathcal{X}^d \times \mathcal{Y}$, we have

$$e_i = y_i - \theta_0 - \boldsymbol{\theta}^T \mathbf{x_i}, \quad i = 1, 2, \ldots, n$$

- Using residual error, we can define different loss functions:

$$\mathcal{L}(\theta_0, \boldsymbol{\theta}) = \sum_{i=1}^{n} \left(y_i - \theta_0 - \boldsymbol{\theta}^T \mathbf{x_i}\right)^2 \qquad \text{Least-squared error (LSE)}$$

$$\mathcal{L}(\theta_0, \boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} \left(y_i - \theta_0 - \boldsymbol{\theta}^T \mathbf{x_i}\right)^2 \qquad \text{Mean-squared error (MSE)}$$

$$\mathcal{L}(\theta_0, \boldsymbol{\theta}) = \sqrt{\frac{1}{n} \sum_{i=1}^{n} \left(y_i - \theta_0 - \boldsymbol{\theta}^T \mathbf{x_i}\right)^2} \qquad \text{Root Mean-squared error (RMSE)}$$

- *One* minimizer for all loss functions.

Linear Regression

Define Loss Function:

- We minimize the following loss function: $\mathcal{L}(\theta_0, \boldsymbol{\theta}) = \frac{1}{2}\sum_{i=1}^{n}\left(y_i - \theta_0 - \boldsymbol{\theta}^T\mathbf{x_i}\right)^2$

- We have an **optimization problem**: find the parameters which minimize the loss function. We write optimization problem (with no constraints) as

$$\underset{\theta_0,\boldsymbol{\theta}}{\text{minimize}} \quad \mathcal{L}(\theta_0, \boldsymbol{\theta}) = \frac{1}{2}\sum_{i=1}^{n}\left(y_i - \theta_0 - \boldsymbol{\theta}^T\mathbf{x_i}\right)^2$$

How to solve?

- Analytically: Determine a critical point that makes the derivtive (if it exists) equal to zero.

- Numerically: Solve optimization using some algorithm that iteratively takes us closer to the critical point minimizing objective function.

Factor $\frac{1}{2}$ is added to make the formulation mathematically more convenient.

Linear Regression

Define Loss Function:

Reformulation:

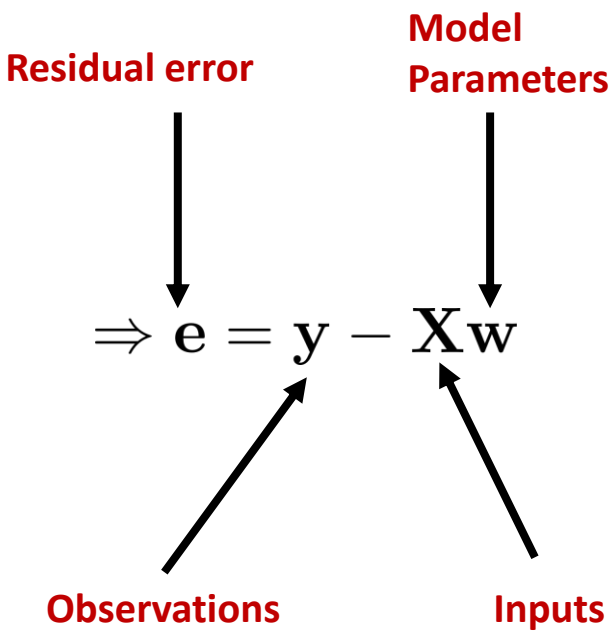
$$\mathcal{L}(\theta_0, \boldsymbol{\theta}) = \frac{1}{2}\sum_{i=1}^{n}\left(y_i - \theta_0 - \boldsymbol{\theta}^T\mathbf{x_i}\right)^2 = \frac{1}{2}\mathbf{e}^T\mathbf{e}$$

Here $\mathbf{e} = [e_1, e_2, \ldots, e_n]^T$ (column vector) where

$$e_i = y_i - \theta_0 - \mathbf{x_i}^T\boldsymbol{\theta}, \quad i = 1, 2, \ldots, n$$

$$\begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} - \theta_0 \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} - \begin{bmatrix} \mathbf{x_1}^T \\ \mathbf{x_2}^T \\ \vdots \\ \mathbf{x_n}^T \end{bmatrix} \boldsymbol{\theta} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} - \underbrace{\begin{bmatrix} 1 & \mathbf{x_1}^T \\ 1 & \mathbf{x_2}^T \\ \vdots & \\ 1 & \mathbf{x_n}^T \end{bmatrix}}_{\mathbf{X}} \begin{bmatrix} \theta_0 \\ \boldsymbol{\theta} \end{bmatrix}$$

$\mathbf{y}$ $\qquad$ $\mathbf{w}$



Consequently:

$$\mathcal{L}(\theta_0, \boldsymbol{\theta}) = \mathcal{L}(\mathbf{w}) = \frac{1}{2}(\mathbf{y} - \mathbf{X}\mathbf{w})^T(\mathbf{y} - \mathbf{X}\mathbf{w})$$

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2}\|(\mathbf{y} - \mathbf{X}\mathbf{w})\|_2^2$$

Linear Regression

Solve Optimization Problem: (Analytical Solution employing Calculus)

$$\underset{\mathbf{w}}{\text{minimize}} \quad \mathcal{L}(\mathbf{w}) = \frac{1}{2}\|(\mathbf{y} - \mathbf{X}\mathbf{w})\|_2^2$$

- *Very beautiful, elegant function we have here!*

We first write the loss function as

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2}(\mathbf{y} - \mathbf{X}\mathbf{w})^T(\mathbf{y} - \mathbf{X}\mathbf{w})$$

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2}(\mathbf{y}^T\mathbf{y} - \mathbf{y}^T\mathbf{X}\mathbf{w} - \mathbf{w}^T\mathbf{X}^T\mathbf{y} + \mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w})$$

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2}(\mathbf{y}^T\mathbf{y} - 2\mathbf{w}^T\mathbf{X}^T\mathbf{y} + \mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w})$$

- To further solve this, let us quickly talk about the concept of a gradient of a function.

Linear Regression

Solve Optimization Problem: (Analytical Solution employing Calculus)

Gradient of a function: Overview

- For a function $f(\mathbf{x})$ that maps $\mathbf{x} \in \mathbf{R}^d$ to $\mathbf{R}$, we define a gradient (direec-tional derivative) with respect to $\mathbf{x}$ as

$$\nabla f(\mathbf{x}) = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_d}\right]^T \in \mathbf{R}^d$$

- Interpretation: Quantifies the rate of change along different directions.

Examples:

- $f(\mathbf{x}) = \mathbf{a}^T\mathbf{x} = \mathbf{x}^T\mathbf{a}$

  $\nabla f(\mathbf{x}) = \mathbf{a}$

- $f(\mathbf{x}) = \mathbf{x}^T\mathbf{x}$

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

- $f(\mathbf{x}) = \mathbf{x}^T\mathbf{P}\mathbf{x}$

  $\nabla f(\mathbf{x}) = 2\mathbf{P}\mathbf{x}$

Linear Regression

Solve Optimization Problem: (Analytical Solution employing Calculus)

We have a loss function: $\mathcal{L}(\mathbf{w}) = \frac{1}{2}(\mathbf{y}^T\mathbf{y} - 2\mathbf{w}^T\mathbf{X}^T\mathbf{y} + \mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w})$

- Take gradient with respect to $\mathbf{w}$ as

$$\nabla\mathcal{L}(\mathbf{w}) = \frac{1}{2}(-2\mathbf{X}^T\mathbf{y} + 2\mathbf{X}^T\mathbf{X}\mathbf{w})$$

- Substituting it equal to zero yields

$$\mathbf{X}^T\mathbf{X}\mathbf{w} = \mathbf{X}^T\mathbf{y}$$

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

- We have determined the weights for which LSE, MSE, RMSE or the norm of the residual is minimized.
- This solution is referred to as least-squared solution as it minimizes the squared error.

Linear Regression

So far and moving forward:

- We assumed that we know the structure of the model, that is, there is a linear relationship between inputs and output.
- Number of parameters = dimension of the feature space + 1 (bias parameter)
- Formulated loss function using residual error.
- Formulated optimization problem and obtain analytical solution.
- Linear regression is one of the models for which we can obtain an analytical solution.
- We will shortly learn an algorithm to solve optimization problem numerically.

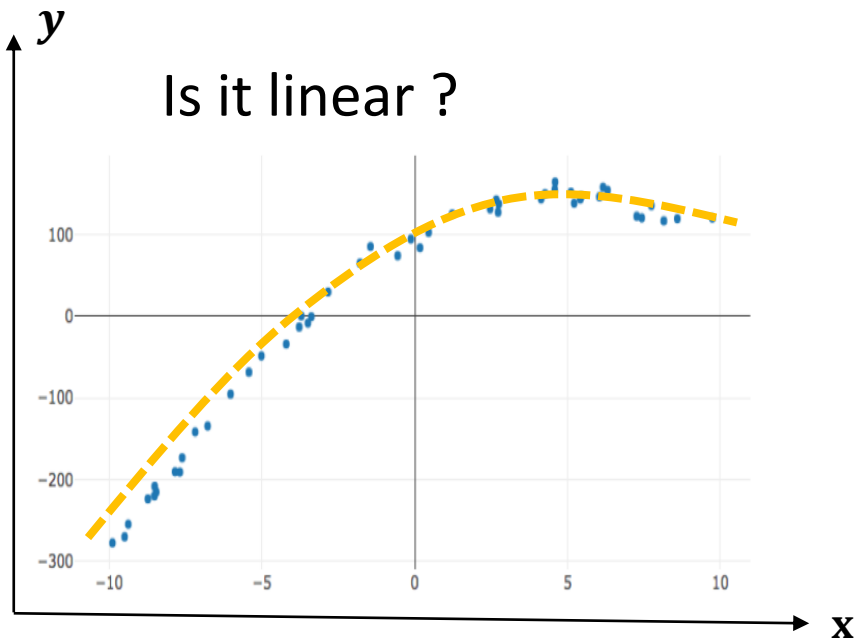
Outline

- Regression Set-up
- Linear Regression
- Polynomial Regression
- Underfitting/Overfitting
- Regularization
- Gradient Descent Algorithm

Polynomial Regression

Overview:

- If the relationship between the inputs and output is not linear, we can use a polynomial to model the relationship.
- We will formulate the polynomial regression model for single feature regression problem.
- Polynomial Regression is often termed as **Non-linear Regression** or **Linear in Parameter Regression.**
- We will also revisit the concept of 'over-fitting'.



Polynomial Regression

Single Feature Regression:

Formulation:

- $d = 1$, input x is a scalar.
- Model is a polynomial function of the input, that is,

$$\hat{f}(x, \boldsymbol{\theta}) = \theta_0 + \theta_1 x + \theta_2 x^2 + \ldots + \theta_M x^M = \sum_{i=0}^{M} \theta_i x^i$$

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_M \end{bmatrix}$$

- M is the degree of plynomial; characterized by $M+1$ coefficients $\theta_0, \theta_1, \ldots, \theta_M$.

- M is the Hyper-Parameter of the model and determines the complexity of the model. For $M = 1$, we have a linear regression.

- We can use linear regression to find these coefficients by formulating the input x and its powers using a vector-valued function given by

$$\boldsymbol{g}(x) = \left[1, x, x^2, \cdots, x^M\right]^T$$

Polynomial Regression

Single Feature Regression:

Formulation:

- With this notation, we can formulate model as $\hat{f}(x, \boldsymbol{\theta}) = \boldsymbol{g}(x)^T \boldsymbol{\theta}$
- Note that the model is linear in terms of parameters due to which Polynomial Regression is termed as Linear in Parameter Regression.
- Note that $\boldsymbol{g}(x)$ can be any function of x. For example, we can have $\boldsymbol{g}(x) = \left[\frac{1}{x}, \sin(2\pi x), x^2, e^x \dots\right]^T$
- For n data points (input, output), we can define residual error in a similar way we computed for linear regression as follows:

$$\begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} - \underbrace{\begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^M \\ 1 & x_2 & x_2^2 & \dots & x_2^M \\ \vdots & \vdots & \vdots & \vdots & \\ 1 & x_n & x_n^2 & \dots & x_n^M \end{bmatrix}}_{\mathbf{X}} \underbrace{\begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_M \end{bmatrix}}_{\boldsymbol{\theta}} \Rightarrow \mathbf{e} = \mathbf{y} - \mathbf{X}\boldsymbol{\theta}$$

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

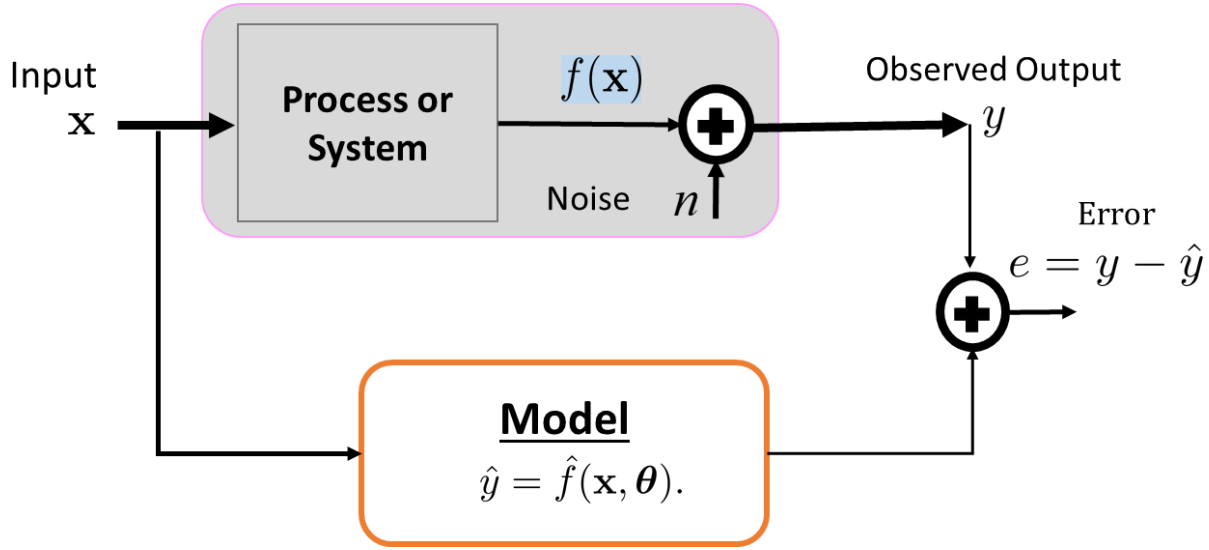
We have seen this before.
&
We are capable to solve this!

Polynomial Regression

Single Feature Regression:

Example (Ref: CB. Section 1.1):

- Model is a polynomial function of degree M.
- If M is not knwown, how do we choose it?

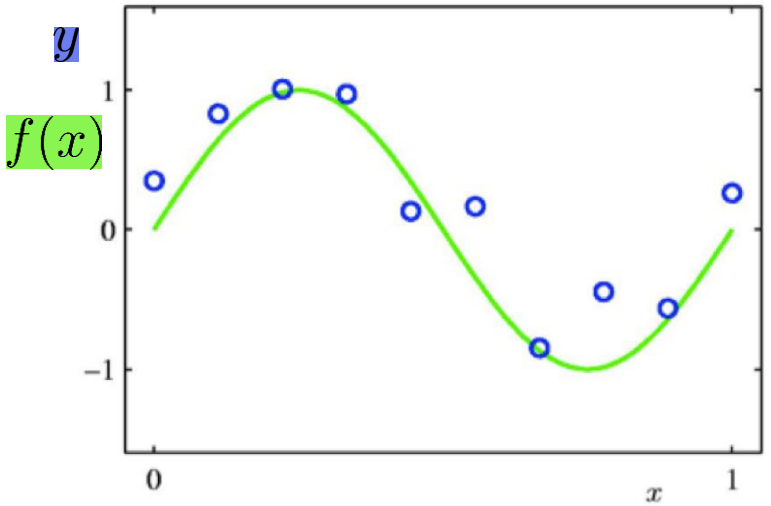


Process $f(x) = \sin(2\pi x)$

Observations $y = f(x) + n$

Model $\hat{f}(x, \boldsymbol{\theta}) = \theta_0 + \theta_1 x + \theta_2 x^2 + \ldots + \theta_i x^M$

- We take $n = 10$.

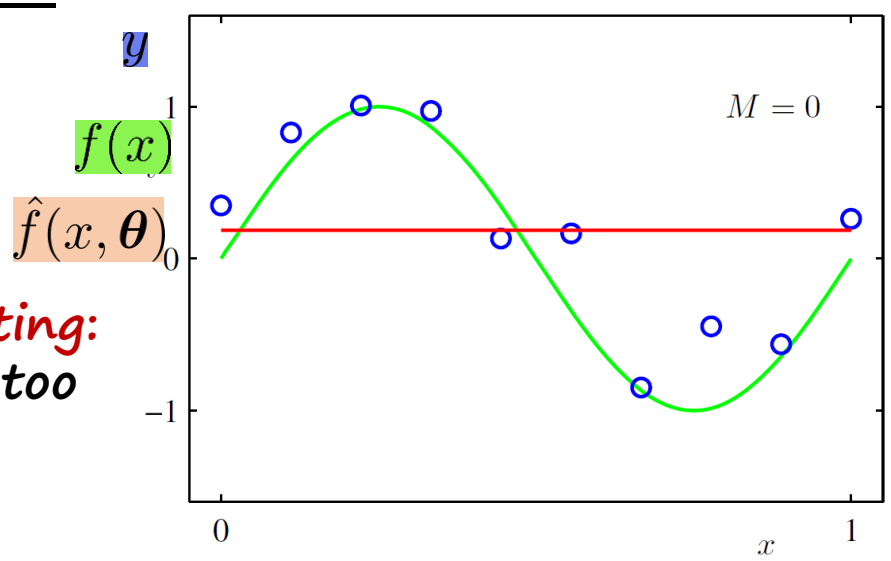


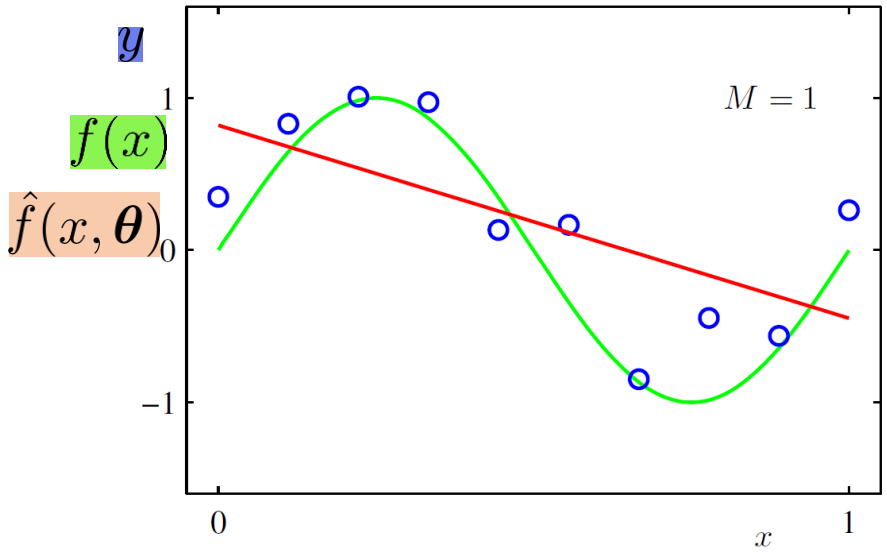
Polynomial Regression

Single Feature Regression: $f(x) = \sin(2\pi x)$

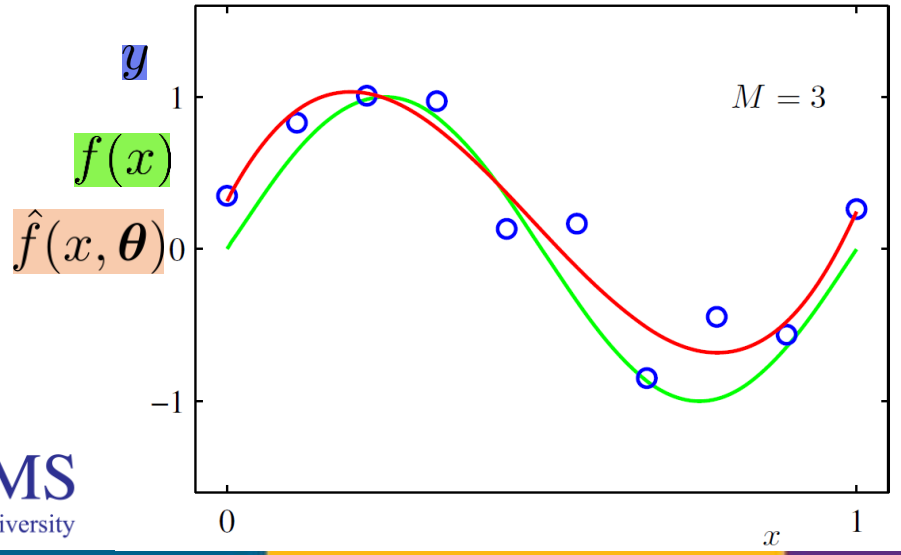
Example:

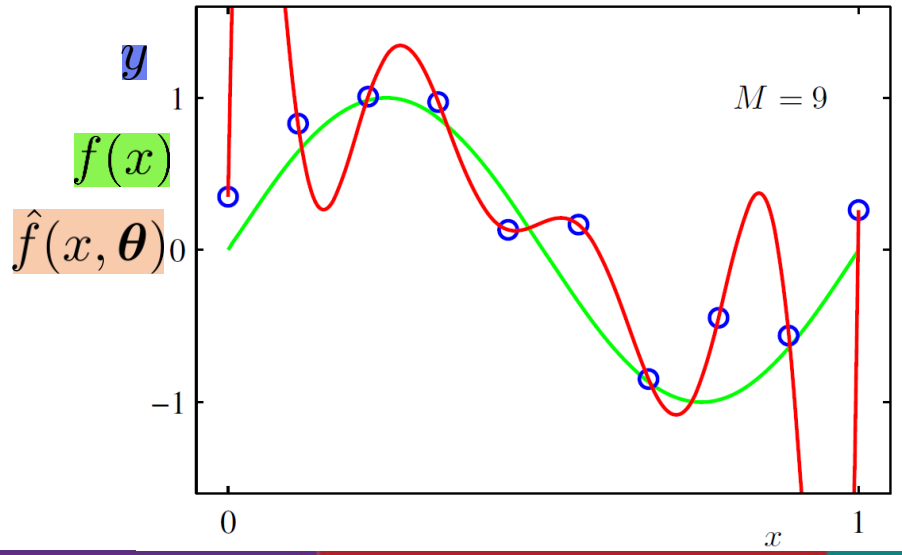
$$\hat{f}(x, \boldsymbol{\theta}) = \theta_0 + \theta_1 x + \theta_2 x^2 + \ldots + \theta_i x^M$$





Underfitting: Model is too simple





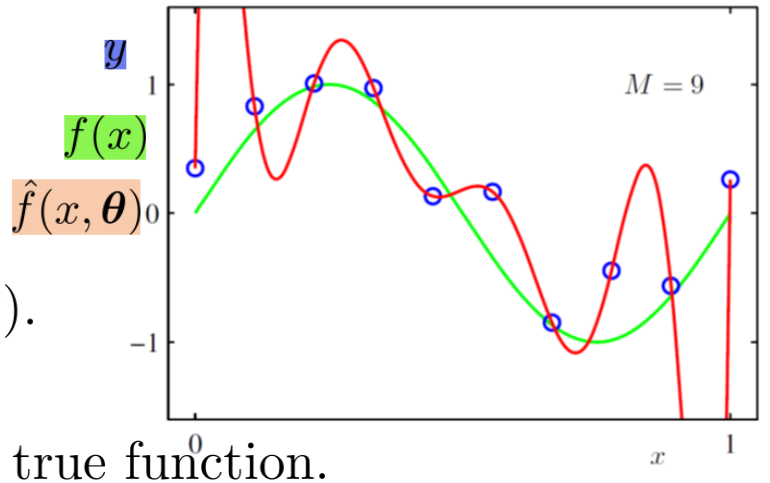
Overfitting: Model is too complex

Polynomial Regression

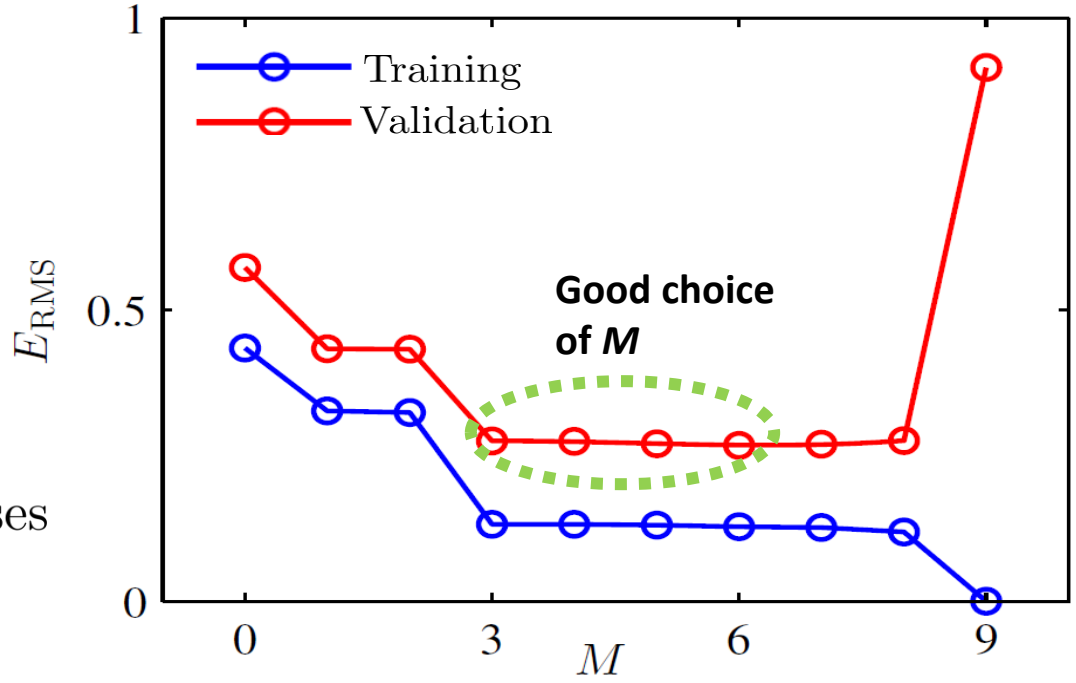
Single Feature Regression:

Example:

- What's happening with the increase in M? *Overfitting*
  - Model is fitting to the data, not the actual true function.
  - For $M = 9$, we have zero residual error, that is, $y = \hat{f}(x, \boldsymbol{\theta})$.
  - Is this a good solution?
  - No! The model is oscillating wildly and is not close to the true function.



- In this toy example, we had informtion about the true function and therefore we can conclude that $M = 9$, is not a good model to fit the data.
- How to choose model order M or How do we tell if a model is overfitting when we do not have knowledge about the true process/function?



Solution 1:

- Recall: Train-Validation Split. Overfitting causes poor generalization performance, that is, large error on the testing or validation data.

Polynomial Regression

Single Feature Regression:

Example:

- Let's pose another question!
- $M = 3$ degree polynomial is a special case of $M = 9$ degree polynomial.
  - Why $M = 9$ gives us poor performance?
- Coefficients magnitude increases with M.
- $M = 3$ solution cannot be recovered from $M = 9$ soltuion by setting the remaining weights equal to zero.
- 10 coefficients are tuned for 10 data-points when $M = 9$.

	$M = 0$,	$M = 1$,	$M = 3$,	$M = 9$
$\boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \\ \theta_5 \\ \theta_6 \\ \theta_7 \\ \theta_8 \\ \theta_9 \end{bmatrix}$	0.19	0.82	0.31	0.35
		-1.27	7.99	232.37
			-25.43	-5321.83
			17.37	48568.31
				-231639.30
				640042.26
				-1061800.52
				1042400.18
				-557682.99
				125201.43

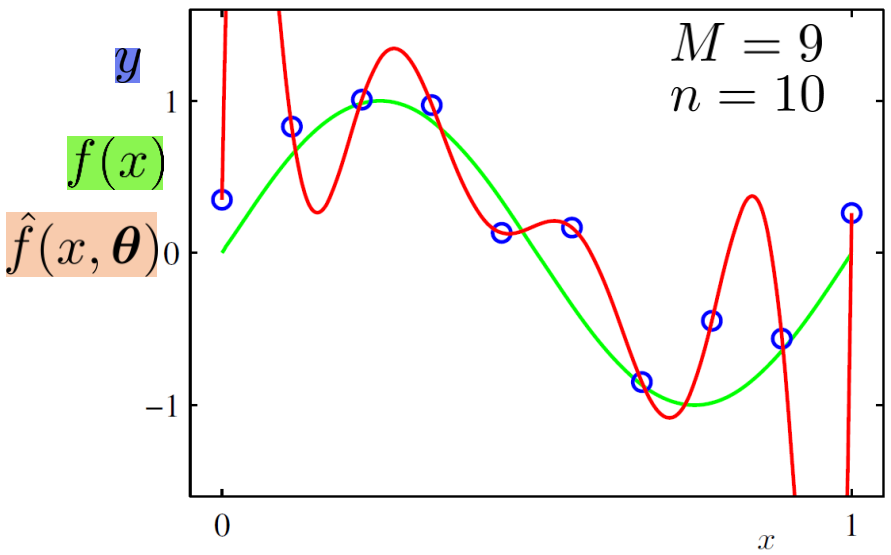
Polynomial Regression

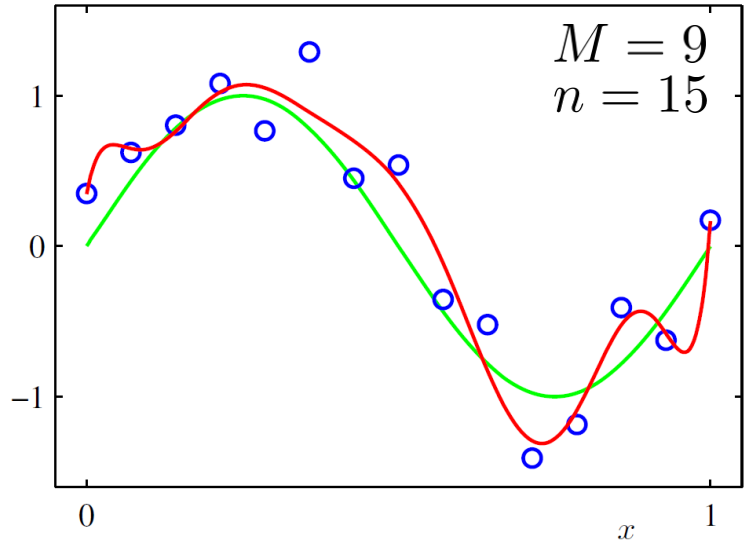
Single Feature Regression:

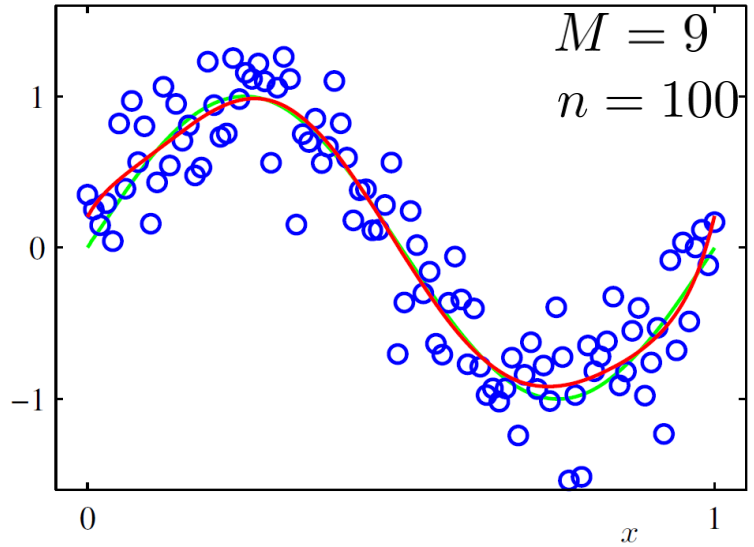
How to Handle Overfitting?

- The polynomial degree M is the hyper-parameter of our model, like we had k in kNN, and controls the complexity of the model.
- If we stick with M=3 model, this is the restriction on the number of parameters.
- We encounter overfitting for M=9 because we do not have sufficient data.

Solution 2: Take more data points to avoid over-fitting.







Solution 3: Regularization

Outline

- Regression Set-up
- Linear Regression
- Polynomial Regression
- Underfitting/Overfitting
- Regularization
- Gradient Descent Algorithm

Regularization

Regularization overview:

- The concept is broad but we will see in the context of linear regression or polynomial regression which we formulated as linear regression.
- Encourages the model coefficients to be small by adding a penalty term to the error.
- We had the loss function of the following form that we minimize to find the coefficients:

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \quad \mathcal{L}(\boldsymbol{\theta}) = \frac{1}{2}\|(\mathbf{y} - \mathbf{X}\boldsymbol{\theta})\|_2^2$$

See linear regression formulation.

- We add a 'penalty term', known as regularizer, in the loss function as

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \quad \mathcal{L}_{\text{reg}}(\boldsymbol{\theta}) = \mathcal{L}(\boldsymbol{\theta}) + \lambda\mathcal{R}(\boldsymbol{\theta})$$

Regularized Loss function

Regularizer

- $\lambda \geq 0$ maintains the trade-off between regularizer and the original loss function as it controls the relative importance of the regulrization term.

Regularization

L^2 Least-squares Regularization – Ridge Regression:

- Since we require to discourage the model coefficients from reaching large values; we can use the following simple regularizer:

$$\mathcal{R}(\boldsymbol{\theta}) = \frac{1}{2}\|\boldsymbol{\theta}\|_2^2$$

Known as L^2 or ℓ^2 penalty

- For this choice, regularized loss function becomes

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \quad \mathcal{L}_{\text{reg}}(\boldsymbol{\theta}) = \frac{1}{2}\|(\mathbf{y} - \mathbf{X}\boldsymbol{\theta})\|_2^2 + \frac{\lambda}{2}\|\boldsymbol{\theta}\|_2^2$$

- This regularization term maintains a trade-off between 'fit of the model to the data' and 'square of norm of the coefficients'.
  - If model is fitted poorly, the first term is large.
  - If coefficients have high values, the second term (penalty term) is large.

- Large λ penalizes coefficient values more.

Intuitive Interpretation: We want to minimize the error while keeping the norm of the coefficients bounded.

Regularization

L^2 Least-squares Regularization – Ridge Regression:

- *Regularized loss function is still quadratic, and we can find closed form solution.*

We have a loss function: $\mathcal{L}_{\text{reg}}(\boldsymbol{\theta}) = \frac{1}{2}\|(\mathbf{y} - \mathbf{X}\boldsymbol{\theta})\|_2^2 + \frac{\lambda}{2}\|\boldsymbol{\theta}\|_2^2$

- Take gradient with respect to $\boldsymbol{\theta}$ as

$$\nabla\mathcal{L}_{\text{reg}}(\boldsymbol{\theta}) = \frac{1}{2}\big(-2\mathbf{X}^T\mathbf{y} + 2\mathbf{X}^T\mathbf{X}\boldsymbol{\theta} + 2\lambda\boldsymbol{\theta}\big)$$

- Substituting it equal to zero yields

$$\mathbf{X}^T\mathbf{X}\boldsymbol{\theta} + \lambda\boldsymbol{\theta} = \mathbf{X}^T\mathbf{y} \;\Rightarrow \big(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}\big)\boldsymbol{\theta} = \mathbf{X}^T\mathbf{y}$$

$$\Rightarrow \boldsymbol{\theta} = \big(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}\big)^{-1}\mathbf{X}^T\mathbf{y}$$

- We have a solution of the ridge regression:

$$\boldsymbol{\theta}(\lambda) = \big(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}\big)^{-1}\mathbf{X}^T\mathbf{y}$$

- $\lambda = 0$, we have non-regularized solution.
- $\lambda = \infty$, the solution is a zero vector.

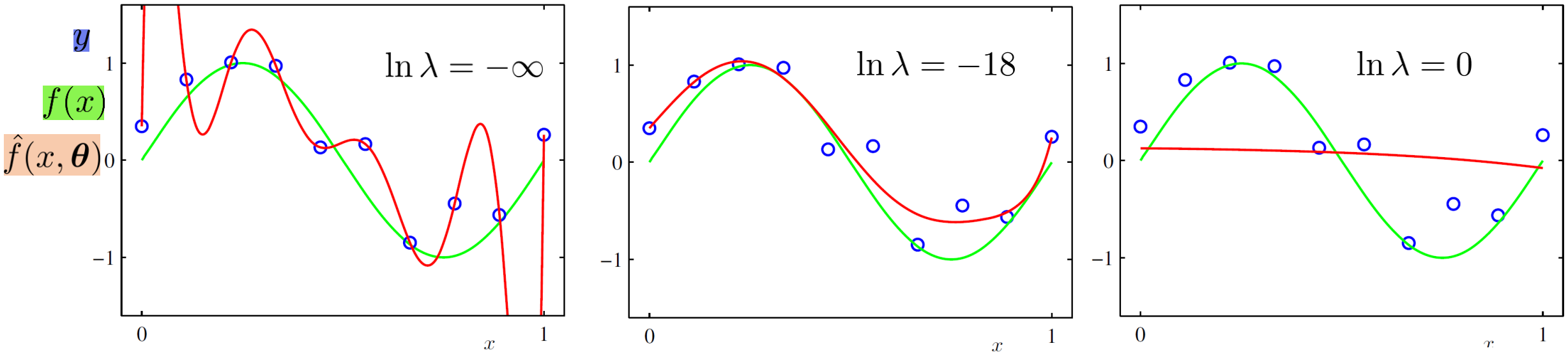
Regularization

L^2 Least-squares Regularization – Ridge Regression:

Example:

- Too small λ: no regularization.
- Too large λ: no weightage to the data.

- In practice, we use very small value of λ and therefore it is convenient to work with $\ln \lambda$ and compute it as $\lambda = e^{\ln \lambda}$.



No regularization

Too much regularization

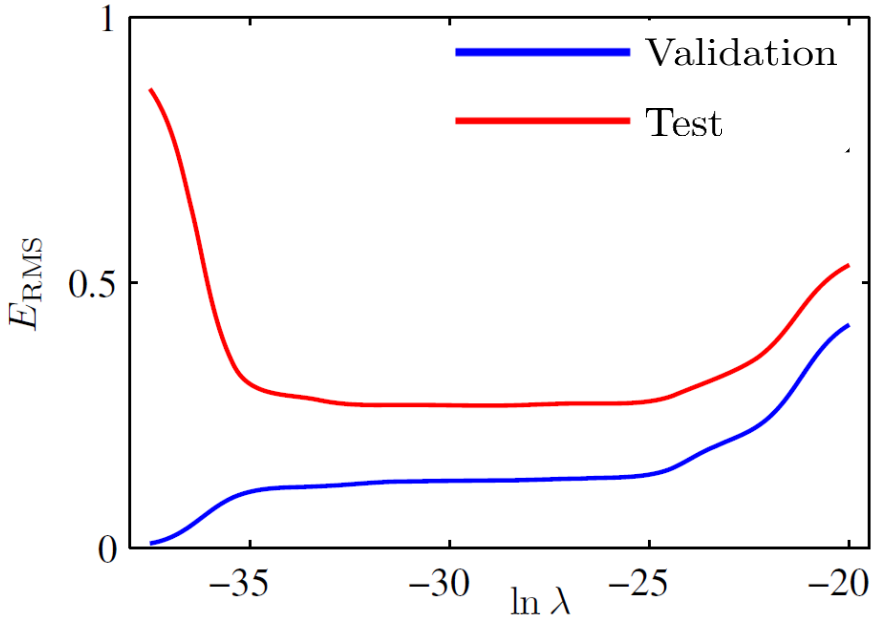
Regularization

L^2 Least-squares Regularization – Ridge Regression:

Example:

- λ restricts the coefficients from exploding as we have included the square of the norm of the coefficients in the loss function being minimized.

	$\ln\lambda = -\infty$	$\ln\lambda = -18$	$\ln\lambda = 0$
θ_0	0.35	0.35	0.13
θ_1	232.37	4.74	-0.05
θ_2	-5321.83	-0.77	-0.06
θ_3	48568.31	-31.97	-0.05
θ_4	-231639.30	-3.89	-0.03
θ_5	640042.26	55.28	-0.02
θ_6	-1061800.52	41.32	-0.01
θ_7	1042400.18	-45.95	-0.00
θ_8	-557682.99	-91.53	0.00
θ_9	125201.43	72.68	0.01



- λ is a hypermater of the model and we learn it in practice using the validation data.

Regularization

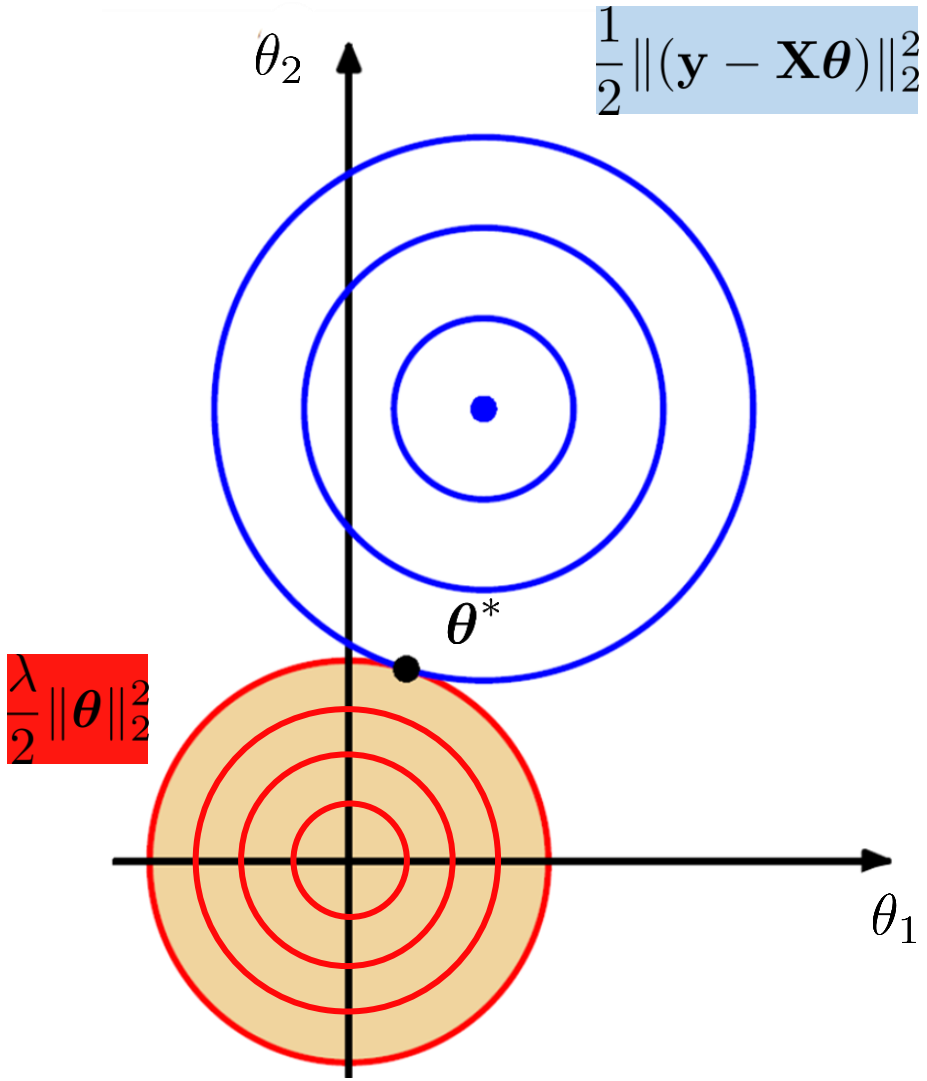
L^2 Least-squares Regularization – Ridge Regression:

Graphical Visualization:

$\boldsymbol{\theta} = [\theta_1, \theta_2]$, we assume we have two coefficients: θ_1 and θ_2.

We have a loss function: $\mathcal{L}_{\text{reg}}(\boldsymbol{\theta}) = \frac{1}{2}\|(\mathbf{y} - \mathbf{X}\boldsymbol{\theta})\|_2^2 + \frac{\lambda}{2}\|\boldsymbol{\theta}\|_2^2$

- Good value of λ helps us in avoiding overfitting.
- Irrelevant features get small but non-zero value in the regularized solution.
- Ideally, we would like to assign zero weight to the irrelevant features.



Regularization

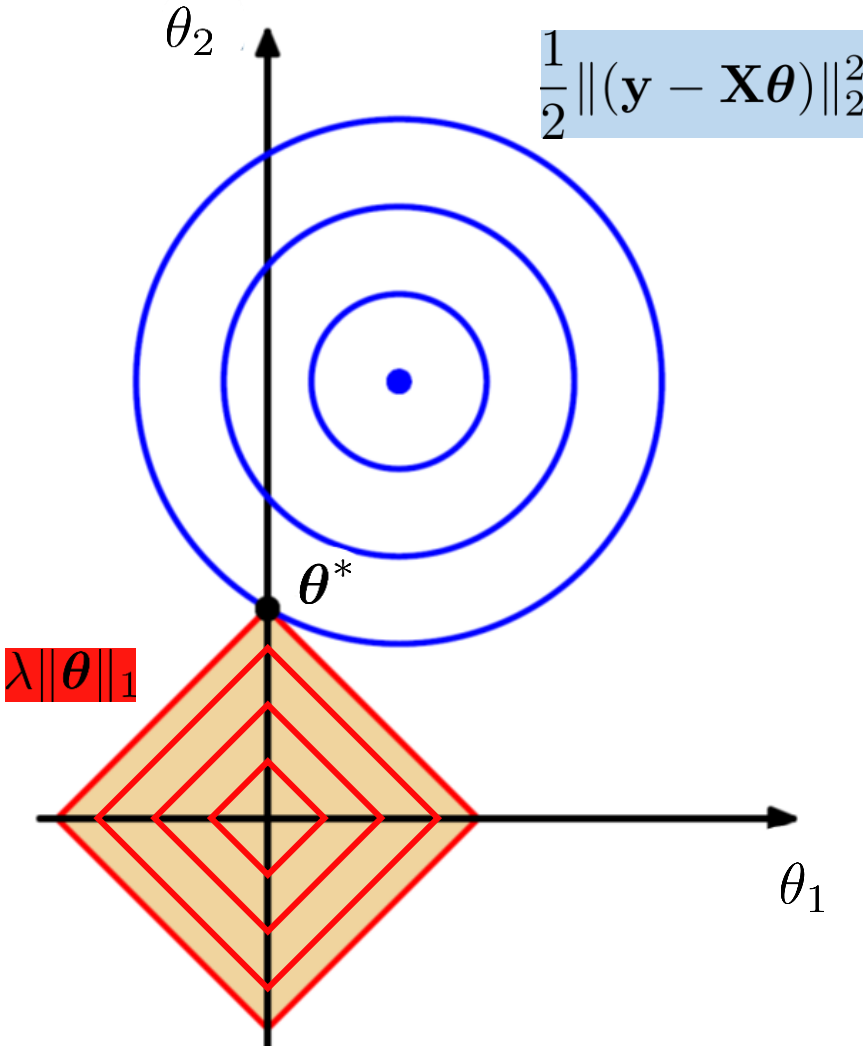
L^1 Least-squares Regularization – Lasso Regression

- Use L^1 or ℓ^1 penalty instead, that is, $\mathcal{R}(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_1 = \sum_i |x_i|$
- For this choice, regularized loss function becomes

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \quad \mathcal{L}_{\text{reg}}(\boldsymbol{\theta}) = \frac{1}{2}\|(\mathbf{y} - \mathbf{X}\boldsymbol{\theta})\|_2^2 + \lambda\|\boldsymbol{\theta}\|_1$$

- This regularization is referred to as least absolute shrinkage and selection operator (Lasso).
- The intersection is at the corners of the diamond.
  - Lasso regression gives us sparse solution.

Graphical Visualization:



Regularization

Elastic Net Regression, L^1 vs L^2

- Ridge: Error + λ times (sum of squares of coefficients)
- Lasso: Error + λ times (sum of absolute values of the coefficients)
- Lasso optimization: computationally expensive than ridge regression.
- Due to the corners included in the solution, regularized solution will have some weights qual to zero.
  - Solution is sparse in general, and is therefore biased.
- **Elastic Net Regression:** Hybrid version; both L_1 and L_2 penalties.

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \quad \mathcal{L}_{\text{reg}}(\boldsymbol{\theta}) = \frac{1}{2}\|(\mathbf{y} - \mathbf{X}\boldsymbol{\theta})\|_2^2 + \lambda_1\|\boldsymbol{\theta}\|_1 + \frac{\lambda_2}{2}\|\boldsymbol{\theta}\|_2^2$$

- Ridge and Lasso are special cases of elastic net regression.
- Combines the strength of both but require tuning of hyperparameters λ_1 and λ_2 using validation data.

Outline

- Regression Set-up
- Linear Regression
- Polynomial Regression
- Underfitting/Overfitting
- Regularization
- Gradient Descent Algorithm

Gradient Descent Algorithm

Optimization and Gradient Descent - Overview

- Optimization refers to finding **optimal** value of your unknown variables under some **constraints** on the variables.
  - Optimal value: usually, maximizing or minimizing the objective function.
  - Constraints: restricting the domain of our variable and are defined by imposing eqality or inequality constraints on the function of the variable.

- An optimization problem of finding a variable $\boldsymbol{\theta}$ is usually formulated as

$$\begin{aligned} &\text{minimize} && f_o(\boldsymbol{\theta}) \\ &\text{subject to} && f_i(\boldsymbol{\theta}) \leq 0, \quad i = 1, 2, \ldots, m \\ &&& h_j(\boldsymbol{\theta}) = 0, \quad , j = 1, 2, \ldots, p \end{aligned}$$

$f_o(\boldsymbol{\theta})$ - Objective function $f_i(\boldsymbol{\theta})$ - Inequality constraint functions $h_j(\boldsymbol{\theta})$ - equality constraint functions

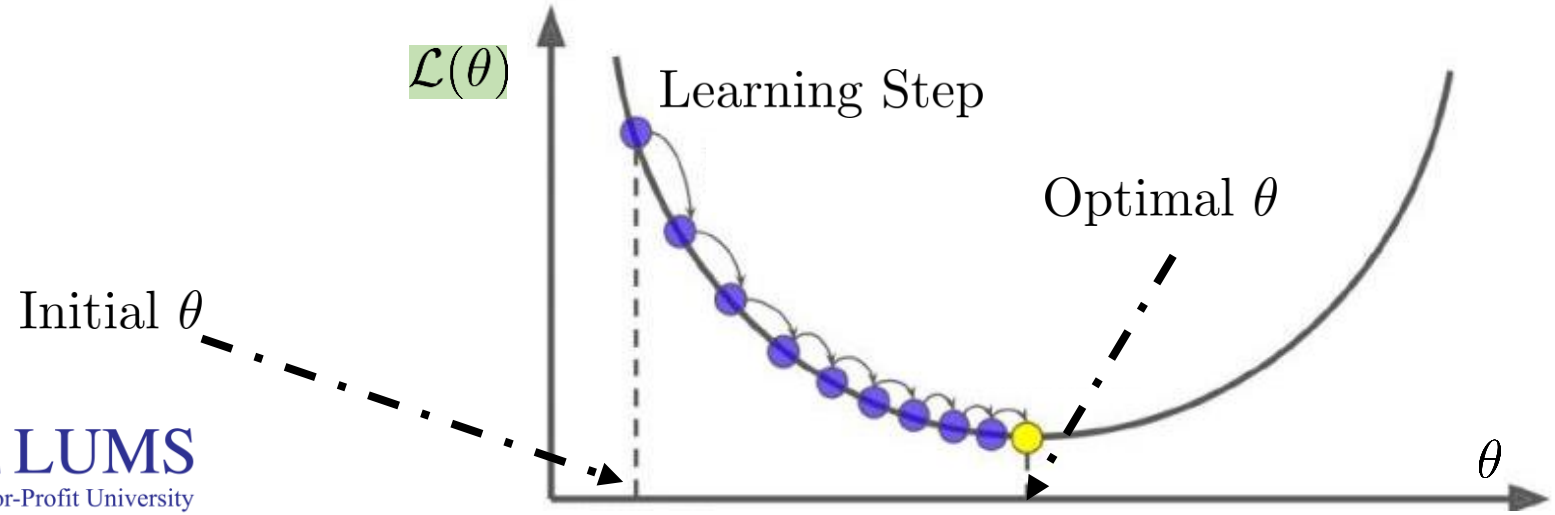
e.g., $\frac{1}{2}\|(\mathbf{y} - \mathbf{X}\boldsymbol{\theta})\|_2^2$

- In ML, various algorithms (e.g., linear regression, neural network etc.) require us to solve an optimization problem.

Gradient Descent Algorithm

Optimization and Gradient Descent - Overview

- To solve the optimization problem, the gradient descent approach or algorithm is the most commonly used method.
- Gradient descent algorithm is best used when tha unknown variables cannot be determined analytically and need to be searched numerically.
- Gradient descent is an iterative algorithm in nature:
  - Initially, chose the coefficients to be something reasonabe (e.g., all zeros).
  - Iteratively update the coefficients in the direction of steepest descent until convergence.
  - Ensures that the new coefficients are better than the previous coefficients.



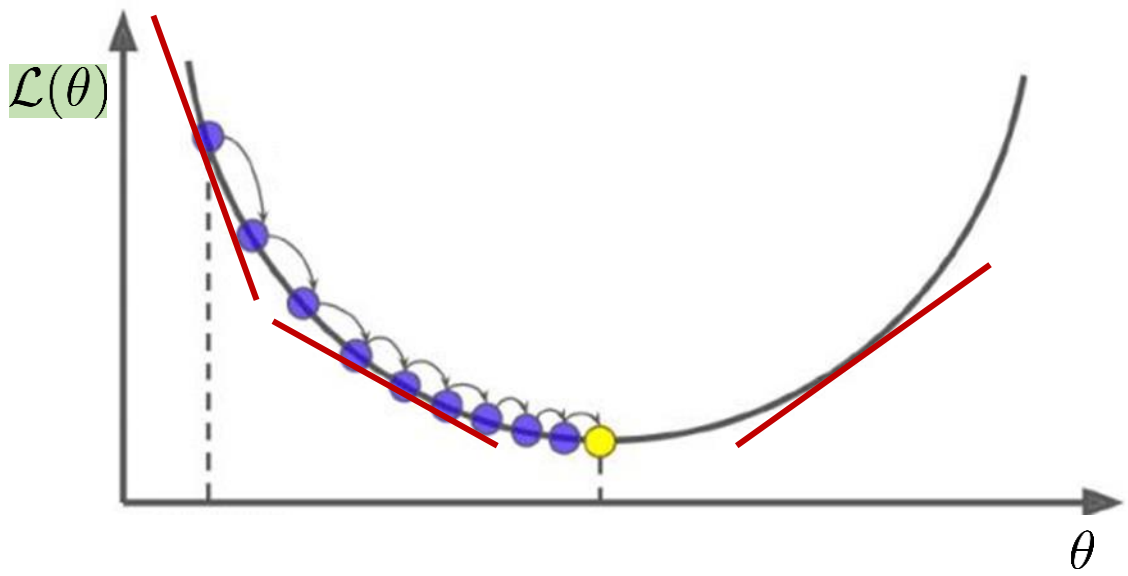
Gradient Descent Algorithm

Formulation:

- Loss function, denoted by $\mathcal{L}(\boldsymbol{\theta})$, required to be minimized. Assume $\boldsymbol{\theta} \in \mathbf{R}^d$.
- Interpretation of $\frac{\partial \mathcal{L}}{\partial \theta_i}$: Rate of change in the loss function with respect to θ_i
  - $\frac{\partial \mathcal{L}}{\partial \theta_i} > 0$: Increasing θ_i increases $\mathcal{L}$
  - $\frac{\partial \mathcal{L}}{\partial \theta_i} < 0$: Increasing θ_i decreases $\mathcal{L}$
- Noting this, the loss function is decreased with the following update:

$$\theta_i \leftarrow \theta_i - \alpha \frac{\partial \mathcal{L}}{\partial \theta_i}, \quad \alpha > 0$$

- This is the essence of gradient descent, that is, the step size in the direction of negative of the derivative.
- α is referred to as step size or learning rate.
  - Too small α: gradient descent can be slow.
  - Too large α: gradient descent can overshoot the minimum and it may fail to converge.



Gradient Descent Algorithm

Algorithm:

Overall:

- Start with some $\boldsymbol{\theta} \in \mathbf{R}^d$ and keep updating to reduce the loss function until we reach the minimum. Repeat until convergence

Pseudo-code:

- Initialize $\boldsymbol{\theta} \in \mathbf{R}^d$.
- Repeat until convergence:

$$\theta_i \leftarrow \theta_i - \alpha \frac{\partial \mathcal{L}}{\partial \theta_i}, \quad \text{for each} \quad i = 1, 2, \ldots, d$$

Equivalently,

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla \mathcal{L}(\boldsymbol{\theta})$$

Note: **Simultaneous update.**

Convergence and Step size:

- We stop updating $\boldsymbol{\theta}$ if $\nabla \mathcal{L}(\boldsymbol{\theta}) = 0$ or difference between the loss function in successive iterations is less than some threshold.
- We can have a constant step size α (typically 0.01, 0.05, 0.001) for each iteration or adjust it adpatively on each iteration.
- Algorithm converges for constant fixed rate as well due to the automatic smaller step size near the optimal solution.

Gradient Descent Algorithm

Linear Regression Case:

- We minimize mean-squared error (MSE) scaled by 1/2 factor:

$$\mathcal{L}(\theta_0, \boldsymbol{\theta}) = \frac{1}{2n} \sum_{i=1}^{n} \left(\theta_0 + \boldsymbol{\theta}^T \mathbf{x_i} - y_i\right)^2$$

- First we take a single feature regression; $\mathbf{x}$ is a scalar, $\mathbf{x_i} \equiv x_i$.

$$\mathcal{L}(\theta_0, \theta_1) = \frac{1}{2n} \sum_{i=1}^{n} \left(\theta_0 + \theta_1 x_i - y_i\right)^2$$

- We define partial derivatives as

$$\frac{\partial \mathcal{L}}{\partial \theta_0} = \frac{1}{n} \sum_{i=1}^{n} (\theta_0 + \theta_1 x_i - y_i) \qquad \frac{\partial \mathcal{L}}{\partial \theta_1} = \frac{1}{n} \sum_{i=1}^{n} (\theta_0 + \theta_1 x_i - y_i)\, x_i$$

Gradient Descent:

- Repeat until convergence:

$$\theta_0 \leftarrow \theta_0 - \alpha \frac{1}{n} \sum_{i=1}^{n} (\theta_0 + \theta_1 x_i - y_i) \qquad \theta_1 \leftarrow \theta_1 - \alpha \frac{1}{n} \sum_{i=1}^{n} (\theta_0 + \theta_1 x_i - y_i)\, x_i$$

Note:

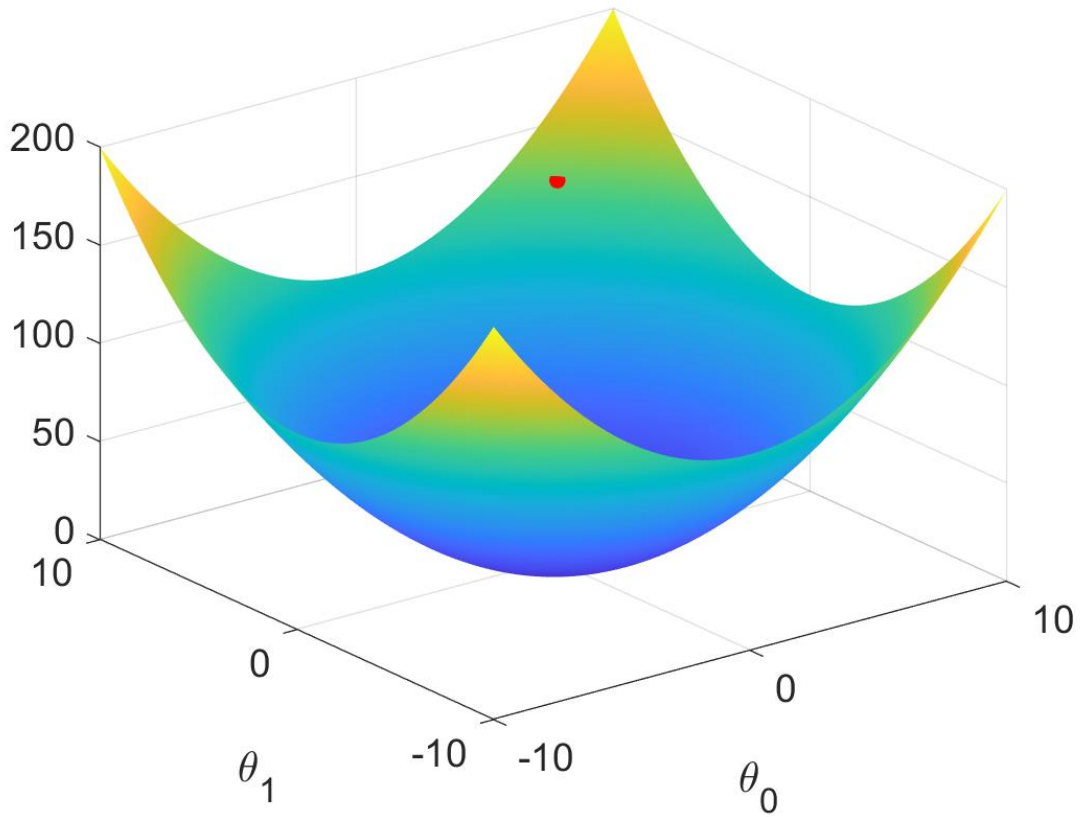
Simultaneous update.

Gradient Descent Algorithm

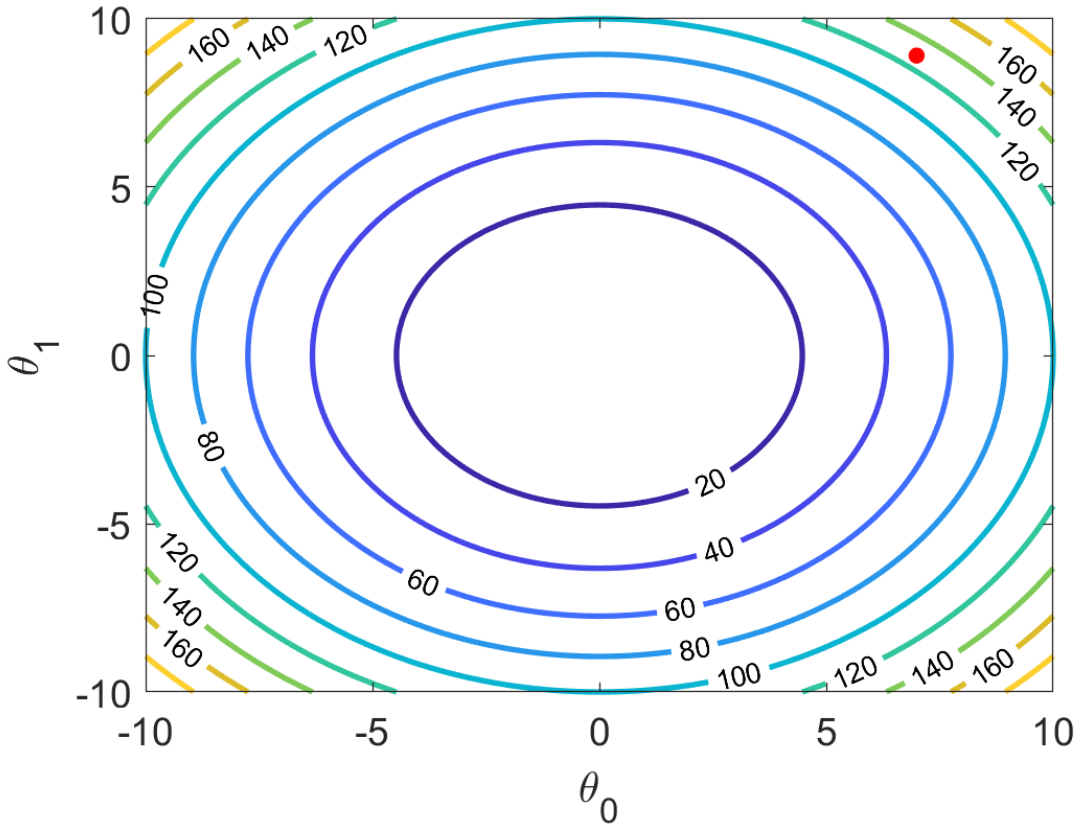
Linear Regression Case:

Visualization:

$$\mathcal{L}(\theta_0, \theta_1)$$



Surface plot



Contour plot

$$\alpha = 0.05,\ 0.2,\ 0.8,\ 1$$

Gradient Descent Algorithm

Linear Regression Case:

- For a multiple feature regression; $\mathbf{x}$ is a vector, $\mathbf{x_i} \in \mathbf{R}^d$.

$$\mathcal{L}(\theta_0, \boldsymbol{\theta}) = \frac{1}{2n} \sum_{i=1}^{n} \left(\theta_0 + \boldsymbol{\theta}^T \mathbf{x_i} - y_i\right)^2$$

- We define partial derivatives as

$$\frac{\partial \mathcal{L}}{\partial \theta_0} = \frac{1}{n} \sum_{i=1}^{n} (\theta_0 + \boldsymbol{\theta}^T \mathbf{x_i} - y_i) \qquad \frac{\partial \mathcal{L}}{\partial \theta_j} = \frac{1}{n} \sum_{i=1}^{n} (\theta_0 + \boldsymbol{\theta}^T \mathbf{x_i} - y_i) \mathbf{x_i}^{(j)}$$

where $\mathbf{x_i}^{(j)}$ denotes the j-th component of $\mathbf{x_i}$.

Gradient Descent:

- Repeat until convergence:

$$\theta_0 \leftarrow \theta_0 - \alpha \frac{1}{n} \sum_{i=1}^{n} (\theta_0 + \boldsymbol{\theta}^T \mathbf{x_i} - y_i) \qquad \boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \frac{1}{n} \sum_{i=1}^{n} (\theta_0 + \boldsymbol{\theta}^T \mathbf{x_i} - y_i) \mathbf{x_i}$$

Note:

Simultaneous update.

Gradient Descent Algorithm

Notes:

- As we have taken all n points for updating at each step, we refer to the algorithm discussed here as **Batch Gradient Descent**.
- We also use the term 'epoch' to refer to one sweep of all the points in the data-set. So far, iteration is same as epoch as we have taken all the points at each step.
- We prefer to use gradient descent also for linear regression despite the fact that we can find the optimal solution analytically. **Why?**
- Gradient descent is easy to implement than the analytical solution.
- Gradient descent is computationally more efficient:
  - Closed-form (direct) solution: $\left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}$
  - Size of $\mathbf{X}^T\mathbf{X}$ is $d \times d$, matrix inversion computational complexity is $\mathcal{O}(d^3)$.
  - Computational complexity of each update of gradient descent is $\mathcal{O}(n\,d)$.
  - $\mathcal{O}(n\,d)$ is better than $\mathcal{O}(d^3)$ when $d \gg 1$.

Stochastic Gradient Descent:

- For large data-sets such that the computation of gradient for all points in the data-set takes too much time, we use stochastic gradient descent.

Gradient Descent Algorithm

Stochastic Gradient Descent (SGD) - Rationale:

- Generalize the formulation by defining a loss function using model $\hat{f}(\mathbf{x_i}, \mathbf{w})$

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2n}\sum_{i=1}^{n}\left(\hat{f}(\mathbf{x_i}, \mathbf{w}) - y_i\right)^2 = \frac{1}{n}\sum_{i=1}^{n} g(\mathbf{x_i}, \mathbf{w}, y_i)$$

where

$$g(\mathbf{x_i}, \mathbf{w}, y_i) = \frac{1}{2}\left(\hat{f}(\mathbf{x_i}, \mathbf{w}) - y_i\right)^2$$

Quantifies the prediction error for a single input.

- In Batch (or Full) Gradient Descent, we update in each iteration as

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla \mathcal{L}(\mathbf{w})$$

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{1}{2n}\sum_{i=1}^{n} \nabla\left(\hat{f}(\mathbf{x_i}, \mathbf{w}) - y_i\right)^2$$

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{1}{n}\sum_{i=1}^{n} \nabla g(\mathbf{x_i}, \mathbf{w}, y_i)$$

- We are computing gradient for all n points.
  - n can be very large in practice.
  - Computationally expensive.

Gradient Descent Algorithm

Stochastic Gradient Descent (SGD):

- Stochastic gradient descent: update using one data point at each iteration

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla g(\mathbf{x_i}, \mathbf{w}, y_i)$$

- Also referred to as incremental or online gradient descent.
- This update tries to approximate the update of batch gradient descent.
- Q: How do we choose i in each iteration?
  - Stochastic selection: uniformly choose the index in each iteration.
  - Cyclic selection: choose $i = 1, 2, \ldots, n, 1, 2, \ldots, n, 1, 2, \ldots$
- Stochastic (random) selection, mostly used in practice, implies that SGD is using an unbiased estimate of the true gradient at each iteration.

Pros:

- Computationally efficient: iteration cost is independent of n.
- True gradient approximation can help in escaping the local minimum.

Gradient Descent Algorithm

SGD for Linear Regression Case:

- Using cyclic selection, we have the following SGD:

- Initialize $\theta_0 \in \mathbf{R}$ and $\boldsymbol{\theta} \in \mathbf{R}^d$.
- Repeat until convergence:

for $i = 1, 2, \ldots n$

$$\theta_0 \leftarrow \theta_0 - \alpha(\theta_0 + \boldsymbol{\theta}^T \mathbf{x_i} - y_i)$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha\,(\theta_0 + \boldsymbol{\theta}^T \mathbf{x_i} - y_i)\mathbf{x_i}$$

} Iteration

} Epoch

end for

- Even with cyclic seleciton, we shuffle the order in which we are using the data points after each epoch. Otherwise, algorithm can get stuck with the sequence of gradient updates that may cancel each other and conseqeuntly hinder learning.
- For online learning when the data points are arriving in a stream, we need to carry out predictions before we have all the data-points. In such a case, we use SGD for learning.

Gradient Descent Algorithm

Mini-batch Stochastic Gradient Descent (SGD) :

Batch Gradient Descent

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{1}{n} \sum_{i=1}^{n} \nabla g(\mathbf{x_i}, \mathbf{w}, y_i)$$

Stochastic Gradient Descent

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla g(\mathbf{x_i}, \mathbf{w}, y_i)$$

- Mini-batch Stochastic Gradient Descent: update using a subset of k data-points.
- From a set $\mathcal{D}$ of n points, we randomly select a subset, denoted by $\mathcal{S} \subseteq \mathcal{D}$ of $k \ll n$ points and use these k points to update the gradient as

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{1}{n} \sum_{i=1}^{k} \nabla g(\mathbf{x_i}, \mathbf{w}, y_i), \quad (\mathbf{x_i}, y_i) \in \mathcal{S}$$

- In one epoch, we divide the data into mini-batches and run mini-batch SGD on each subset iteratively.

Feedback: Questions or Comments?

Email: zubair.khalid@lums.edu.pk