# Week 2: Linear Regression

Yunwen Lei

School of Computer Science, University of Birmingham

In this week, we will study linear regression. First, we will give a basic formulation of linear regression problems. Then we will show how to solve the linear regression problems by a closed-form solution. This is based on the fundamental concept of gradients. Therefore, we recall basic concepts of derivative and gradient. We also show how the simple linear regression method can be extended naturally to nonlinear regression methods such as polynomial regression.

## 1 Linear Regression

A first question is why linear regression is worth studying. Here we give some reasons.

- Linear regression (least squares) is at least 200 years old going back to Legendre and Gauss.
- It is a fundamental and theoretically sound method. Its properties are well studied by researchers and are easy to understand.
- Although simple, linear models can be very powerful in practice. This is especially the case if we have many features. Note each feature gives some information about the learning problem. Then a linear model with many features already has a strong representation ability.
- More complex models require understanding linear regression. For example, deep neural networks have several layers. Each layer consists of a linear model and a nonlinear activation function. Therefore, we need to first understand linear models before studying deep neural networks.
- Linear regression has closed form solution. This means that we can represent the solution via mathematical equations. Then, linear regression problems are easy to solve.
- Many key notions of machine learning can be introduced. For example, by taking a look of linear regression we can introduce underfitting, overfitting and regularization.

**A Toy Example: Commute Time**. We first give a practical example to motivate linear regression. Suppose we want to predict the commute time to UoB.


There are several variables useful for our prediction. For simplicity, we only consider two variables.

- Distance to UoB (if the distance is large then the commute time is large)
- Day of the week (we only consider whether it is a weekday or a weekend, a weekday mean a crowed traffic)

We can collect the Data into a table as follows. In this case, we have 5 training examples and each training examples has two input attributes and one output.

| dist(km) | day | commute time (min) |
|---|---|---|
| 2.7 | 1 | 25 |
| 4.1 | 1 | 33 |
| 1.0 | 0 | 15 |
| 5.2 | 1 | 45 |
| 2.8 | 0 | 22 |

day = 1 if weekday, 0 if weekend

We aim to find a function $f : \mathbb{R}^2 \mapsto \mathbb{R}$ such that

$$f(\text{dist}, \text{day}) \approx \text{commute time}$$

## 1.1 Problem Setup

Dataset: $n$ input/output pairs ($n$ is the sample size and $d$ is the number of features, or dimension)

$$S = \{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(n)}, y^{(n)})\}$$

- $\mathbf{x}^{(i)} \in \mathbb{R}^d$ is the "input" for the $i$-th data point as a feature vector with $d$ elements. E.g. (dist, day) in our toy example ($d = 2$)
- $y^{(i)} \in \mathbb{R}$ is the "output" for the $i$-th data point. E.g. Commute Time in our toy example

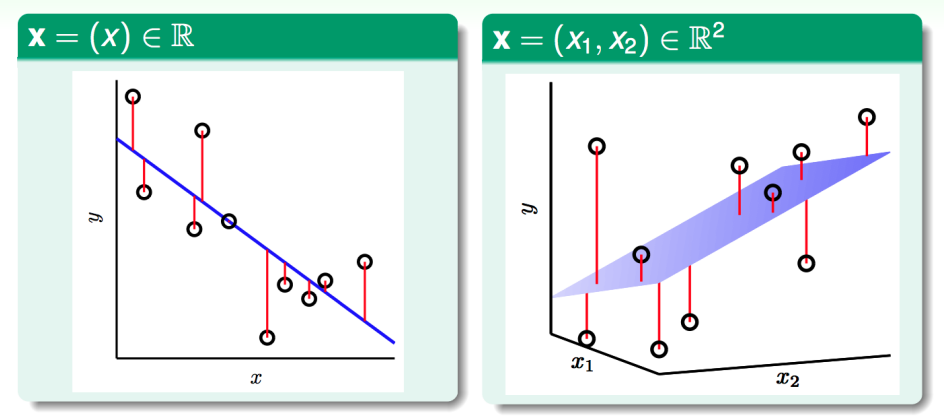
Regression Task: find a model, i.e., a function $f : \mathbb{R}^d \mapsto \mathbb{R}$ such that the predicted output $f(X)$ is close to the true output $Y$. In this way, the function $f$ can be used for prediction.

Linear Model: a linear regression model has the form

$$f(\mathbf{x}) = w_0 + w_1 x_1 + \ldots + w_d x_d$$

- bias (intercept): $w_0$
- weight parameter: $w_1, \ldots, w_d$
- feature: $x_i$ is the $i$-th component of $\mathbf{x} \in \mathbb{R}^d$.

As you see, the bias and weights are parameters which determine the function $f$. $f$ is a linear function of the parameters. The problem of finding a linear model $f$ is equivalent to the construction of parameters $w_0, \ldots, w_d$ from the data.




Linear regression: find linear function with small discrepancy!

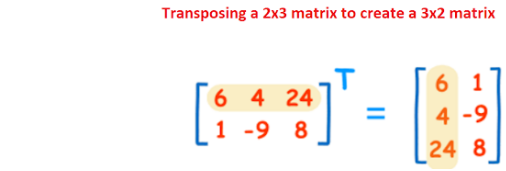
**Remark 1.** Here we use superscripts and subscripts. We use superscripts to mean different examples and use subscripts to mean different features of a single example. For example, $\mathbf{x}_j^{(i)}$ means the $j$-th feature of the $i$-th example. Other books may use different notations.

## 1.2 Recap: Matrices and vectors

We recall some basic operators on matrices and vectors.

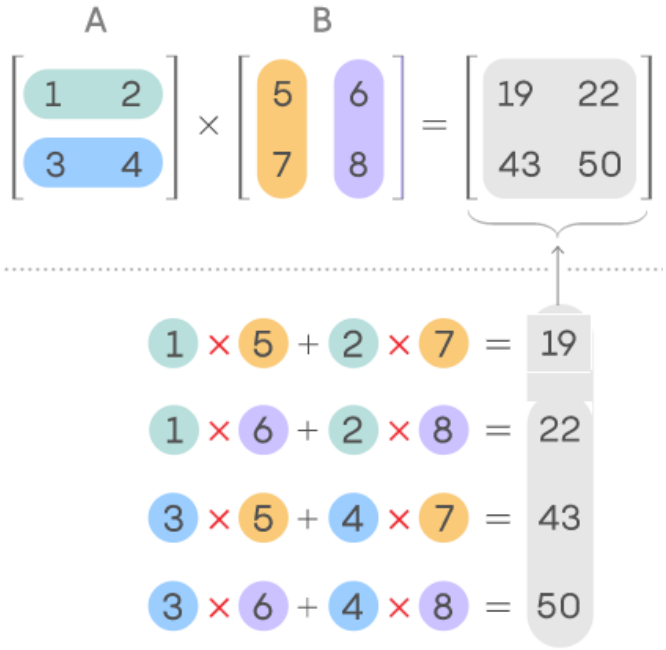
- For a matrix $A \in \mathbb{R}^{m\times n}$, $A_{i,j}$ denotes the element in the $i$-th row and $j$-th column.
- matrix transpose: $A^\top$ is defined by

$$A^\top_{i,j} = A_{j,i}, \quad A^\top \in \mathbb{R}^{n\times m}$$




- matrix multiplication: If $B \in \mathbb{R}^{n\times r}$, then

$$(AB)_{i,j} = \sum_{k=1}^{n} A_{i,k}B_{k,j}, \quad AB \in \mathbb{R}^{m\times r}$$




- For two vectors $\mathbf{u} = (u_1, \ldots, u_m)^\top, \mathbf{v} = (v_1, \ldots, v_m)^\top \in \mathbb{R}^m$

**vector addition**

$$\mathbf{u}+\mathbf{v} = \begin{pmatrix} u_1 + v_1 \\ \vdots \\ u_m + v_m \end{pmatrix}$$

**dot product**

$$\mathbf{u}^\top\mathbf{v} = (u_1, \ldots, u_m)\begin{pmatrix} v_1 \\ \vdots \\ v_m \end{pmatrix} = \sum_{i=1}^{m} u_i v_i$$

**Hadamard product**

$$\mathbf{u} \odot \mathbf{v} = \begin{pmatrix} u_1 v_1 \\ \vdots \\ u_m v_m \end{pmatrix}$$

## 1.3 Linear Model : Adding a feature for bias term

In our definition of a linear model, we need to consider the bias and other weight parameters separately. This is not quite convenient. This drawback can be overcome by adding another feature of "1" to the input. That is, we can add 1 to get an expanded feature vector

| dist(km) | day | commute time |
|---|---|---|
| $x_1$ | $x_2$ | $y$ |
| 2.7 | 1 | 25 |
| 4.1 | 1 | 33 |
| 1.0 | 0 | 15 |
| 5.2 | 1 | 45 |
| 2.8 | 0 | 22 |

$\Longleftrightarrow$

| one | dist(km) | day | commute time |
|---|---|---|---|
| $x_0$ | $x_1$ | $x_2$ | $y$ |
| 1 | 2.7 | 1 | 25 |
| 1 | 4.1 | 1 | 33 |
| 1 | 1.0 | 0 | 15 |
| 1 | 5.2 | 1 | 45 |
| 1 | 2.8 | 0 | 22 |

**Definition 1** (Linear model)**.** We do not need to consider specially the bias for a linear model

$$f(\mathbf{x}) = w_0 + w_1 x_1 + \ldots + w_d x_d = \underbrace{(w_0, w_1, w_2, \ldots, w_d)}_{:=\mathbf{w}^\top} \underbrace{\begin{pmatrix} 1 \\ \mathbf{x} \end{pmatrix}}_{:=\bar{\mathbf{x}}} = \mathbf{w}^\top \bar{\mathbf{x}}.$$

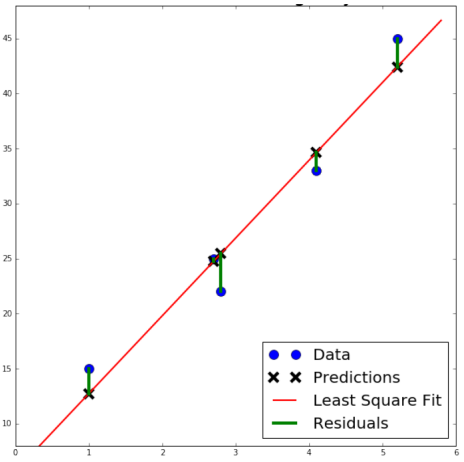
For brevity, we do not consider the bias and set $\mathbf{x}$ as the expanded feature vector, i.e., $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x}$!

**Remark 2.** When we mention a vector, we always mean a column vector. That is why we need to take a transpose $\mathbf{w}^\top$ to get a row vector in the above equation.

## 1.4 Performance Measure

When we have a model $f$, we are interested in its performance in the prediction. This performance can be quantified by a cost function $C(\mathbf{w}) : \mathbb{R}^d \mapsto \mathbb{R}$.

- Intuitively, $C(\mathbf{w})$ quantifies the error in the predictions for a given parameter $\mathbf{w}$. The smaller the better. Below we show how to construction $C(\mathbf{w})$ for linear regression problems.
- The "**residual**" on the $i$-th data point can be defined as
$$e^{(i)} := y^{(i)} - \mathbf{w}^\top \mathbf{x}^{(i)},$$
where measures the difference between the true output $y^{(i)}$ and the predicted output $\mathbf{w}^\top \mathbf{x}^{(i)}$ of the model $\mathbf{w}$ on the $i$-th example
- The following mean square error (MSR) $C$ takes into account the errors for all $n$ data points
$$C(\mathbf{w}) = \frac{1}{2n} \sum_{i=1}^{n} \left(y^{(i)} - \mathbf{w}^\top \mathbf{x}^{(i)}\right)^2.$$
It is clear that $C(\mathbf{w})$ considers the average performance of the model on all examples.
- By squaring the residual, we
  - ignore the sign of the residuals
  - penalize large residuals more (if $e^{(i)} > 1$). That is, if the residual on an example is 100, the loss on that example would be 10000. If the residual on an example is 0.1, the loss on that example would be 0.01. 0.01 is nothing as compared to 10000.




Find the parameter $\mathbf{w}$ which minimizes the loss $C(\mathbf{w})$!

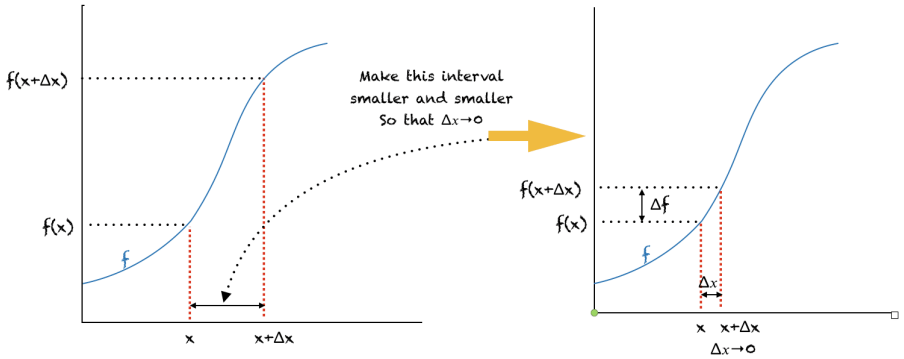
# 2 Gradients

## 2.1 Derivative and Gradients

**Definition 2** (Derivative)**.** The derivative of a function $f : \mathbb{R} \mapsto \mathbb{R}$ is the rate of change of $f$

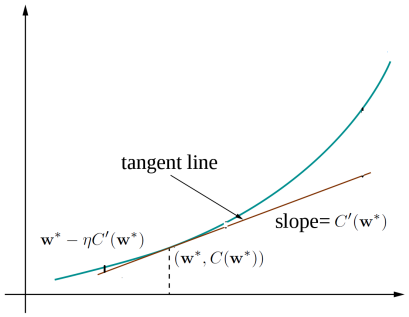
$$f'(x) = \frac{df}{dx} = \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x} = \lim_{\Delta x \to 0} \frac{\Delta f}{\Delta x}.$$

We can illustrate this definition through the following figure




- The derivative of $f$ at $x$ is the slope of the tangent line to the graph of $f$ at $(x, f(x))$
- The tangent line is the best linear approximation of the function near that input value (this approximation will become more accurate if $\bar{x}$ gets closer to $x$)

$$f(\bar{x}) \approx \underbrace{f(x) + f'(x)(\bar{x} - x)}_{\text{tangent line}}.$$




According to the definition, one can check the following computation on the derivatives of basic functions.

**Proposition 1** (Derivative of Basic Functions)**.**

- *Constant function:* $a' = 0$.
- *Linear function:* $(ax)' = a$
- *Quadratic function:* $(ax^2)' = 2ax$
- *Reciprocal function:* $(1/x)' = -1/x^2$
- *exponential function:* $\exp(x)' = \exp(x)$
- *logarithmic function:* $(\log x)' = 1/x$

The derivative is defined for a univariate function. If we consider a multivariate function $f : \mathbb{R}^d \mapsto \mathbb{R}$, we need to consider the partial derivative.

**Definition 3** (Partial derivative)**.** The partial derivative of a **multivariate** function $f(x_1, \ldots, x_d)$ in the direction of variable $x_i$ at $\mathbf{x} = (x_1, \ldots, x_d)$ is

$$\frac{\partial f(x_1, \ldots, x_d)}{\partial x_i} = \lim_{h \to 0} \frac{f(\ldots, x_{i-1}, x_i + h, x_{i+1}, \ldots) - f(x_1, \ldots, x_i, \ldots, x_d)}{h}$$

- Intuitively, $\frac{\partial f}{\partial x_i}$ the derivative of a univariate function

$$g(x_i) := f(x_1, \ldots, x_i, \ldots, x_d),$$

where all variables except $x_i$ are fixed as constants.

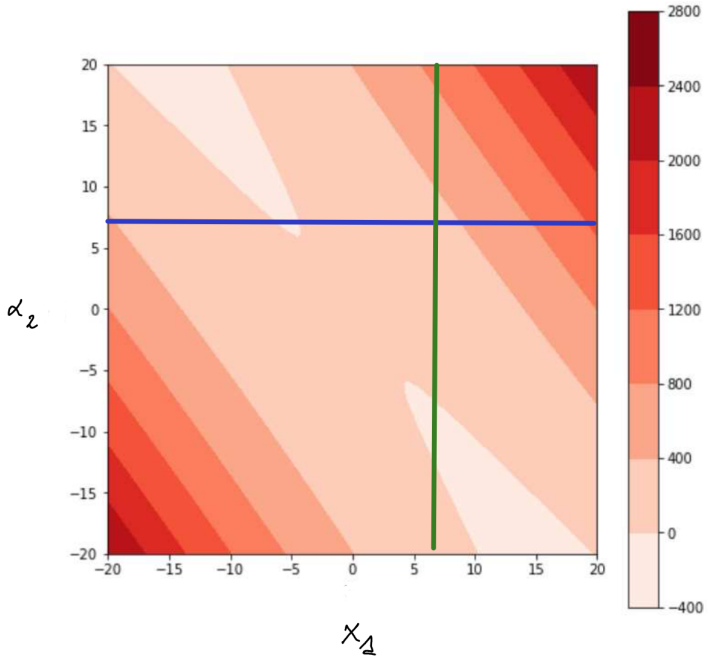
- Example: $f(x_1, x_2) = 2x_1^2 + x_2^2 + 3x_1x_2 + 4$. Then

$$\frac{\partial f}{\partial x_1} = \frac{\partial(2x_1^2 + 3x_1x_2)}{\partial x_1} = 4x_1 + 3x_2$$
$$\frac{\partial f}{\partial x_2} = \frac{\partial(x_2^2 + 3x_1x_2)}{\partial x_2} = 2x_2 + 3x_1.$$

Below we give the geometric interpretation of partial derivative. Here we use color to denote the function value.

- $\frac{\partial f}{\partial x_1}$ is the rate of change of $f$ along dimension $x_1$ (i.e., blue line)
- $\frac{\partial f}{\partial x_2}$ is the rate of change of $f$ along dimension $x_2$ (i.e., green line)

In more details, if we consider $\frac{\partial f}{\partial x_1}$, we fix the variable $x_2$ and view $x_1 \mapsto f(x_1, x_2)$ as a univariate function of $x_1$. $\frac{\partial f}{\partial x_1}$ is then the derivative of this univariate function.




For a multivariate function $f : \mathbb{R}^d \mapsto \mathbb{R}$, we have $d$ partial derivatives. We can collect these $d$ partial derivatives into a vector, which becomes the gradient.

**Definition 4** (Gradient)**.** Let $f : \mathbb{R}^d \mapsto \mathbb{R}$. The gradient of $f$ with respect to $\mathbf{x} \in \mathbb{R}^d$ is defined as

$$\nabla f(\mathbf{x}) = \begin{pmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} \\ \vdots \\ \frac{\partial f(\mathbf{x})}{\partial x_d} \end{pmatrix}.$$

Each component of the gradient measures the change rate of the function value w.r.t. that component. Therefore, the gradient is a vector measuring the change rate of the function value w.r.t. all the components. Analogous to derivatives, we can use gradient to build a linear function to approximate the nonlinear function $f$, i.e.,

$$f(\mathbf{x}') \approx f(\mathbf{x}) + (\mathbf{x}' - \mathbf{x})^\top \nabla f(\mathbf{x}) = f(\mathbf{x}) + \sum_{i=1}^{d} (x_i' - x_i) \frac{\partial f(\mathbf{x})}{\partial x_i}. \tag{2.1}$$

Gradient is a key concept in optimization. The approximation will be more accurate if $\mathbf{x}'$ gets more closer to $\mathbf{x}$. Below we give some examples to show how to compute the gradient of a linear function and a quadratic function. We leave one example as an exercise.

**Example 1** (Linear function)**.** Let $f(\mathbf{x}) = \mathbf{a}^\top \mathbf{x}$, where $\mathbf{a} = (a_1, \ldots, a_d)^\top$. In this case, the gradient is

$$\nabla(\mathbf{a}^\top \mathbf{x}) = \begin{pmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} \\ \frac{\partial f(\mathbf{x})}{\partial x_2} \\ \vdots \\ \frac{\partial f(\mathbf{x})}{\partial x_d} \end{pmatrix} = \begin{pmatrix} \frac{\partial}{\partial x_1} a_1 x_1 + \frac{\partial}{\partial x_1} \sum_{j \neq 1} a_j x_j \\ \frac{\partial}{\partial x_2} a_2 x_2 + \frac{\partial}{\partial x_2} \sum_{j \neq 2} a_j x_j \\ \vdots \\ \frac{\partial}{\partial x_d} a_d x_d + \frac{\partial}{\partial x_d} \sum_{j \neq d} a_j x_j \end{pmatrix} = \begin{pmatrix} a_1 \\ \vdots \\ a_d \end{pmatrix} = \mathbf{a}. \tag{2.2}$$

**Example 2** (Quadratic function)**.** If $f(\mathbf{x}) = \mathbf{x}^\top A \mathbf{x} = \sum_{i,j=1}^{d} a_{i,j} x_i x_j$, where

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,d} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,d} \\ \vdots & \vdots & \ldots & \vdots \\ a_{d,1} & a_{d,2} & \cdots & a_{d,d} \end{pmatrix}, \quad \text{prove } \nabla\left(\mathbf{x}^\top A \mathbf{x}\right) = A\mathbf{x} + A^\top \mathbf{x}. \tag{2.3}$$

In particular, if $A$ is symmetric in the sense of $A = A^\top$, then $\nabla(\mathbf{x}^\top A \mathbf{x}) = 2A\mathbf{x}$.

## 2.2 Unconstrained Minimization

Gradient is a key concept in optimization.

**Definition 5** (Unconstrained minimization)**.** Given an objective function $C : \mathbb{R}^d \mapsto \mathbb{R}$, we want to solve the following optimization problem

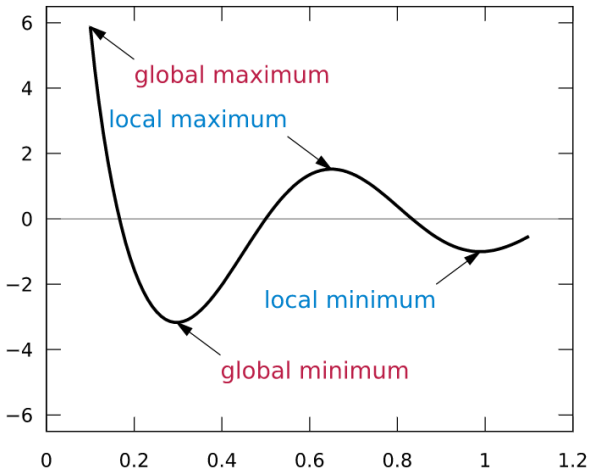
$$\min_{\mathbf{w} \in \mathbb{R}^d} C(\mathbf{w})$$

- $\mathbf{w}^*$ is a Global Minimum Point if

$$C(\mathbf{w}) \geq C(\mathbf{w}^*) \quad \forall \mathbf{w} \in \mathbb{R}^d$$

- $\mathbf{w}^*$ is a Local Minimum Point if there exists $\epsilon > 0$ such that

$$C(\mathbf{w}) \geq C(\mathbf{w}^*) \quad \text{for all } \mathbf{w} \text{ within distance } \epsilon \text{ of } \mathbf{w}^*.$$




Roughly speaking, a global minimum point is a point with the smallest function value over the whole region. A local minimum point means that we can find a neighbourhood such that it attains the smallest function value in that neighbourhood.

We aim to find a minimum of the cost function $C$. A result useful to this aim is the first-order necessary optimality This result shows that we only need to consider those points with vanishing gradients.

**Theorem 2** (First-order Necessary Optimality Condition)**.** *If* $\mathbf{w}^*$ *is a local minimum of a differentiable function* $C$*, then*

$$\nabla C(\mathbf{w}^*) = 0. \tag{2.4}$$

*We say* $\mathbf{w}^*$ *satisfying Eq.* (2.4) *a stationary point.*

Indeed, one can show that if $\nabla C(\mathbf{w}^*) \neq 0$, we can move along the direction $-\nabla C(\mathbf{w}^*)$ to get a smaller function value (Eq. (2.1) with $x' = \mathbf{w}^* - \eta \nabla C(\mathbf{w}^*)$ and $x = \mathbf{w}^*$)

$$\begin{aligned} C(\mathbf{w}^* - \eta \nabla C(\mathbf{w}^*)) &\approx C(\mathbf{w}^*) + \left(\mathbf{w}^* - \eta \nabla C(\mathbf{w}^*) - \mathbf{w}^*\right)^\top \nabla C(\mathbf{w}^*) \\ &= C(\mathbf{w}^*) - \eta \nabla C(\mathbf{w}^*)^\top \nabla C(\mathbf{w}^*) < C(\mathbf{w}^*). \end{aligned}$$

According to the first-order approximation in Eq. (2.1), the above approximation will get closer and closer if $\eta$ approaches to 0.

## 2.3 Closed-form Solution of Linear Regression

Now we use the first-order necessary condition to solve the regression problems. For simplicity, we first consider the one-dimensional case, i.e., $d = 1$. In this case, we know

$$C(w) = \frac{1}{2n} \sum_{i=1}^{n} (x^{(i)} w - y^{(i)})^2 = \frac{1}{2n} \sum_{i=1}^{n} \Big( \underbrace{(x^{(i)})^2 w^2}_{\text{quadratic}} - \underbrace{2 y^{(i)} x^{(i)} w}_{\text{linear}} + \underbrace{(y^{(i)})^2}_{\text{constant}} \Big)$$

It then follows that

$$C'(w) = \frac{1}{2n} \sum_{i=1}^{n} \left( 2(x^{(i)})^2 w - 2 y^{(i)} x^{(i)} \right) = \frac{1}{n} \sum_{i=1}^{n} (x^{(i)})^2 w - \frac{1}{n} \sum_{i=1}^{n} y^{(i)} x^{(i)}.$$

According to the first-order optimality condition, we know the optimal $w^*$ satisfies

$$C'(w^*) = 0 \Longrightarrow \frac{1}{n} \sum_{i=1}^{n} (x^{(i)})^2 w^* = \frac{1}{n} \sum_{i=1}^{n} y^{(i)} x^{(i)}$$

It then follows that

$$w^* = \frac{\sum_{i=1}^{n} y^{(i)} x^{(i)}}{\sum_{i=1}^{n} (x^{(i)})^2}.$$

Then we can consider a general case with $d \in \mathbb{N}$. To this aim, we first give the matrix form for least squares regression. Recall

$$C(\mathbf{w}) = \frac{1}{2n} \sum_{i=1}^{n} ({\mathbf{x}^{(i)}}^\top \mathbf{w} - y^{(i)})^2, \quad \mathbf{x}^{(i)} = \begin{pmatrix} \mathbf{x}_1^{(i)} \\ \vdots \\ \mathbf{x}_d^{(i)} \end{pmatrix}$$

We introduce ($X$ is an $n \times d$ matrix: the $i$-th row represents the $i$-th example, the $j$-th column represents the $j$-th feature)

$$X = \begin{pmatrix} (\mathbf{x}^{(1)})^\top \\ \vdots \\ (\mathbf{x}^{(n)})^\top \end{pmatrix} \in \mathbb{R}^{n \times d}, \mathbf{y} = \begin{pmatrix} y^{(1)} \\ \vdots \\ y^{(n)} \end{pmatrix} \in \mathbb{R}^n \Longrightarrow X\mathbf{w} - \mathbf{y} = \begin{pmatrix} (\mathbf{x}^{(1)})^\top \\ \vdots \\ (\mathbf{x}^{(n)})^\top \end{pmatrix} \mathbf{w} - \begin{pmatrix} y^{(1)} \\ \vdots \\ y^{(n)} \end{pmatrix} = \begin{pmatrix} (\mathbf{x}^{(1)})^\top \mathbf{w} - y^{(1)} \\ \vdots \\ (\mathbf{x}^{(n)})^\top \mathbf{w} - y^{(n)} \end{pmatrix}$$

It follows that

$$\begin{aligned} (X\mathbf{w} - \mathbf{y})^\top (X\mathbf{w} - \mathbf{y}) &= \left( (\mathbf{x}^{(1)})^\top \mathbf{w} - y^{(1)}, \cdots, (\mathbf{x}^{(n)})^\top \mathbf{w} - y^{(n)} \right) \begin{pmatrix} (\mathbf{x}^{(1)})^\top \mathbf{w} - y^{(1)} \\ \vdots \\ (\mathbf{x}^{(n)})^\top \mathbf{w} - y^{(n)} \end{pmatrix} \\ &= \sum_{i=1}^{n} ({\mathbf{x}^{(i)}}^\top \mathbf{w} - y^{(i)})^2 = 2nC(\mathbf{w}) \end{aligned}$$

and (note $(X\mathbf{w})^\top = \mathbf{w}^\top X^\top$)

$$\begin{aligned} C(\mathbf{w}) &= \frac{1}{2n}(X\mathbf{w}-\mathbf{y})^\top(X\mathbf{w}-\mathbf{y}) = \frac{1}{2n}(\mathbf{w}^\top X^\top - \mathbf{y}^\top)(X\mathbf{w}-\mathbf{y}) \\ &= \frac{1}{2n}\Big(\mathbf{w}^\top X^\top X\mathbf{w} - \mathbf{w}^\top X^\top \mathbf{y} - \mathbf{y}^\top X\mathbf{w} + \mathbf{y}^\top\mathbf{y}\Big) \\ &= \frac{1}{2n}\Big(\mathbf{w}^\top X^\top X\mathbf{w} - 2\mathbf{w}^\top X^\top \mathbf{y} + \mathbf{y}^\top\mathbf{y}\Big), \end{aligned}$$

where we have used $\mathbf{w}^\top X^\top \mathbf{y} = (\mathbf{w}^\top X^\top \mathbf{y})^\top = \mathbf{y}^\top X\mathbf{w}$ since $\mathbf{w}^\top X^\top \mathbf{y}$ is a real number.

Now we are ready to derive the closed-form solution in the general case. The above objective function is

$$C(\mathbf{w}) = \frac{1}{2n}\Big(\underbrace{\mathbf{w}^\top X^\top X\mathbf{w}}_{\text{quadratic}} - 2\underbrace{\mathbf{w}^\top X^\top \mathbf{y}}_{\text{linear}} + \underbrace{\mathbf{y}^\top\mathbf{y}}_{\text{constant}}\Big)$$

We compute the gradient by Eq. (2.2), (2.3)

$$\nabla C(\mathbf{w}) = \frac{1}{2n}\Big(2X^\top X\mathbf{w} - 2X^\top \mathbf{y}\Big) \tag{2.5}$$

By the first-order necessary condition, we know that optimal $\mathbf{w}^*$ satisfies

$$\nabla C(\mathbf{w}^*) = \frac{1}{n}\Big(X^\top X\mathbf{w}^* - X^\top \mathbf{y}\Big) = 0 \Longrightarrow (X^\top X)\mathbf{w}^* = X^\top \mathbf{y} \quad \text{(normal equation)}$$

If $X^\top X$ is invertible, we get

$$\mathbf{w}^* = (X^\top X)^{-1}X^\top \mathbf{y}, \tag{2.6}$$

where $(X^\top X)^{-1}$ denotes the inverse of the matrix $X^\top X$. The prediction on the training examples becomes

$$\hat{\mathbf{y}} = X\mathbf{w}^* = \begin{pmatrix} (\mathbf{x}^{(1)})^\top \mathbf{w}^* \\ \vdots \\ (\mathbf{x}^{(n)})^\top \mathbf{w}^* \end{pmatrix} \Longrightarrow \hat{\mathbf{y}} = X(X^\top X)^{-1}X^\top \mathbf{y}$$

**Example 3.** We can apply the above discussion to our toy example with the following dataset

| one | dist(km) | weekday? | commute time (min) |
|---|---|---|---|
| $x_0$ | $x_1$ | $x_2$ | $y$ |
| 1 | 2.7 | 1 | 25 |
| 1 | 4.1 | 1 | 33 |
| 1 | 1.0 | 0 | 15 |
| 1 | 5.2 | 1 | 45 |
| 1 | 2.8 | 0 | 22 |

We have $n = 5, d = 3$ and so we get (this can be computed by python)

$$\mathbf{y} = \begin{pmatrix} 25 \\ 33 \\ 15 \\ 45 \\ 22 \end{pmatrix}, \quad X = \begin{pmatrix} 1 & 2.7 & 1 \\ 1 & 4.1 & 1 \\ 1 & 1.0 & 0 \\ 1 & 5.2 & 1 \\ 1 & 2.8 & 0 \end{pmatrix}, \quad \mathbf{w} = \begin{pmatrix} w_0 \\ w_1 \\ w_2 \end{pmatrix}$$

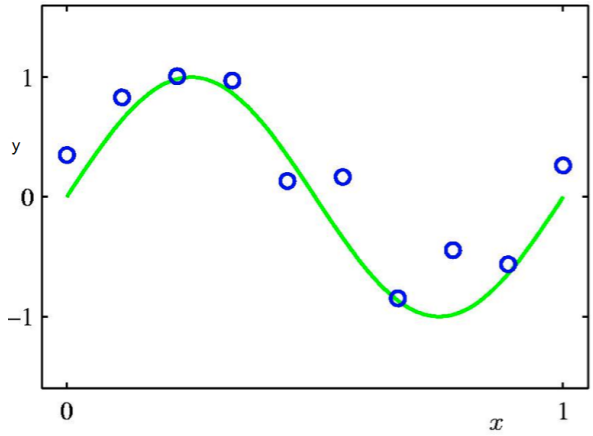
By the closed-form solution, we get

$$\mathbf{w}^* = \begin{pmatrix} 6.09 \\ 6.53 \\ 2.11 \end{pmatrix} \quad \hat{\mathbf{y}} = \begin{pmatrix} 25.84 \\ 34.99 \\ 12.62 \\ 42.17 \\ 24.38 \end{pmatrix} \Longrightarrow \hat{\mathbf{e}} = \begin{pmatrix} -0.84 \\ -1.99 \\ 2.38 \\ 2.82 \\ -2.38 \end{pmatrix}$$

# 3 Polynomial Regression

## 3.1 Fitting Polynomial Models

In the previous analysis, we only consider linear models. However, linear models can have limited representation power. For example, if the true relationship between the output and the input is highly nonlinear, we cannot get a good performance by using linear models. Suppose we want to model the following data




It is clear that the input-output relationship is nonlinear! In this case, we need to consider nonlinear models. A basic nonlinear model is a low-degree polynomial, which leads to polynomial regression problems with the function of the following form

$$f(x) = w_0 + w_1 x + w_2 x^2 + \cdots + w_M x^M, \tag{3.1}$$

where $M \in \mathbb{N}$ is the degree of the polynomial. To fit the polynomial function from the dataset, we need to compute $w_0, w_1, \ldots, w_M$.

*Do we need to derive a new algorithm to this aim?*

The answer is no! Indeed, we can apply our knowledge in linear regression to get a polynomial function from the dataset. The key observation is that while the polynomial function in Eq. (3.1) is nonlinear w.r.t. the variable $x$, it is a linear function w.r.t. the tunable variable $w_0, \ldots, w_M$. Since we only need to fit the parameters $w_0, \ldots, w_M$, this problem is similar to fitting the coefficients in linear regression. To illustrate this, let us fit a polynomial of degree 3, i.e., $M = 3$. In this case, we can define the feature map

$$\phi(x) = \begin{pmatrix} 1 \\ x \\ x^2 \\ x^3 \end{pmatrix} \begin{matrix} \longrightarrow \text{1st feature} \\ \longrightarrow \text{2nd feature} \\ \longrightarrow \text{3rd feature} \\ \longrightarrow \text{4st feature.} \end{matrix}$$

That is, $\phi$ maps a univariate $x$ to a vector with 4 components. The polynomial regression model in Eq. (3.1) becomes a linear model w.r.t. the new feature

$$f(x) = \mathbf{w}^\top \phi(x) = w_0 + w_1 x + w_2 x^2 + w_3 x^3$$

In this way, we transform a **univariate nonlinear** problem to a **multivariate linear** problem. All of the derivations and algorithms so far in this lecture remain exactly the same. The difference is that we replace the original data matrix $X$ by a new data matrix $\bar{X} \in \mathbb{R}^{n \times 4}$

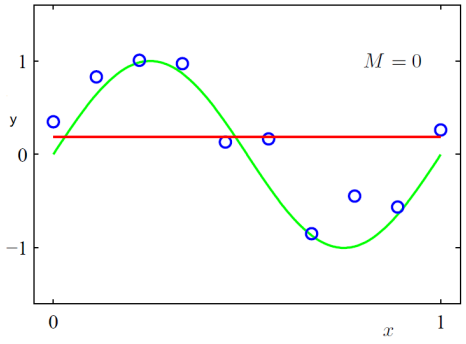
$$X = \begin{pmatrix} (\mathbf{x}^{(1)})^\top \\ (\mathbf{x}^{(2)})^\top \\ \vdots \\ (\mathbf{x}^{(n)})^\top \end{pmatrix} \mapsto \begin{pmatrix} (\phi(x^{(1)}))^\top \\ (\phi(x^{(2)}))^\top \\ \vdots \\ (\phi(x^{(n)}))^\top \end{pmatrix} = \begin{pmatrix} 1 & x^{(1)} & (x^{(1)})^2 & (x^{(1)})^3 \\ 1 & x^{(2)} & (x^{(2)})^2 & (x^{(2)})^3 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x^{(n)} & (x^{(n)})^2 & (x^{(n)})^3 \end{pmatrix} := \bar{X}.$$

Analogous to Eq. (2.6), we get $\mathbf{w}^* = (\bar{X}^\top \bar{X})^{-1} \bar{X}^\top \mathbf{y}$ and the prediction

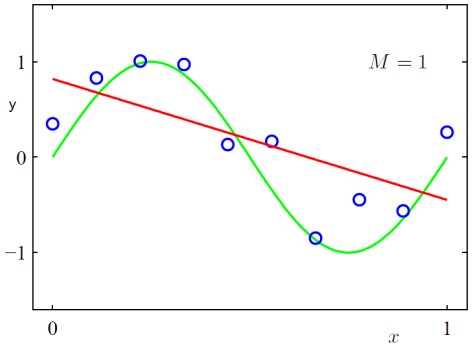
$$\hat{\mathbf{y}} = \bar{X}\mathbf{w}^* = \bar{X}(\bar{X}^\top \bar{X})^{-1} \bar{X}^\top \mathbf{y}.$$

Below we use the above general method to develop polynomials of degrees $0, 1, 3, 9$ to fit the sample.

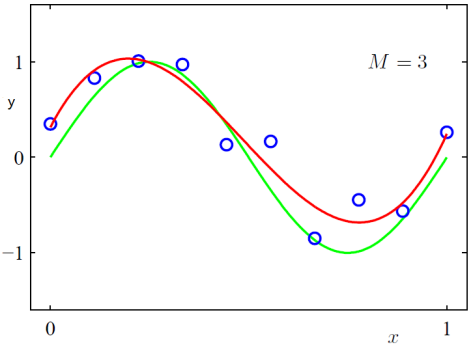
$y = w_0$



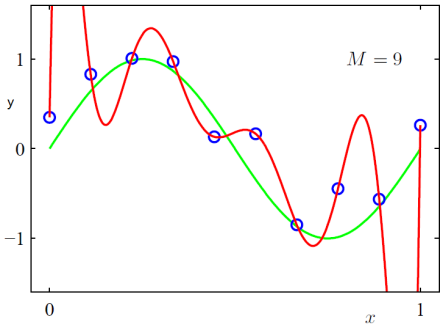

$y = w_0 + w_1 x$




$y = w_0 + w_1 x + w_2 x^2 + w_3 x^3$




$y = w_0 + w_1 x + w_2 x^2 + w_3 x^3 + \ldots + w_9 x^9$

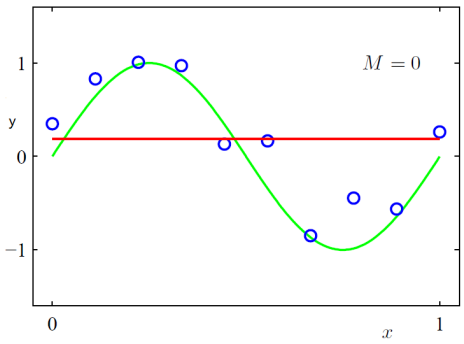



## 3.2 Regularization

According to the above figures, we see different behavior of polynomial models on training examples.

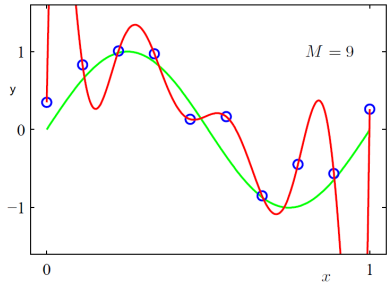
Underfitting: the model is too simple — does not fit the data (bad behavior on training examples and therefore bad behavior on testing examples)

$y = w_0$




Overfitting: the model is too complex — fits perfectly, does not generalize to new data (good behavior on training examples but bad behavior on testing examples)!

$$y = w_0 + w_1 x + w_2 x^2 + w_3 x^3 + \ldots + w_9 x^9$$




Intuitively, underfitting happens because the model has limited representation power (e.g., a linear model can not represent nonlinear data). Overfitting happens because the model is too complex and then the estimation of model parameters would be very difficult. Ideally, we want to get a model with a good behavior on training examples and meanwhile not complex for good generalization. One way to handle this is to encourage the weights to be small (this way no feature will have too much influence on prediction). This is called regularization.

**Definition 6** (Regularized Least Square Regression)**.** Given dataset $S = \{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(n)}, y^{(n)})\}$ and a regularization parameter $\lambda > 0$, find a model to minimize

$$C(\mathbf{w}) = \underbrace{\frac{1}{2n}\sum_{i=1}^{n}\left(y^{(i)} - \mathbf{w}^\top \mathbf{x}^{(i)}\right)^2}_{\text{fitting to data}} + \underbrace{\frac{\lambda}{2}\|\mathbf{w}\|_2^2}_{\text{regularizer}}$$

As you can see, the objective function has two components: the data-fitting term and the regularizer. To minimize the data-fitting term we aim to get a model with a good behavior on training examples. To minimize the regularizer, we want to get a model with simplicity for generalization to unseen data. These two terms are contradictory in general: minimizing data-fitting term asks a complex model and minimizing the regularizer asks for a simple model. We can balance these two terms by choosing an appropriate regularization parameter. If $\lambda$ is large, then we focus on simple models. If $\lambda$ is small, then we focus on complex models.

**Definition 7** (p-norm)**.** For a vector $\mathbf{v} = (v_1, \ldots, v_d) \in \mathbb{R}^d$, the $p$-norm is (2-norm is the Euclidean norm)

$$\|\mathbf{v}\|_p = \left(|v_1|^p + |v_2|^p + \cdots + |v_d|^p\right)^{1/p}.$$

**Theorem 3.** *The solution of the regularized least squares regression is ($\mathbb{I}_n \in \mathbb{R}^{n\times n}$ is the identity matrix)*

$$\mathbf{w}^* = \left(\frac{1}{n}(X^\top X) + \lambda \mathbb{I}\right)^{-1}\left(\frac{1}{n}X^\top \mathbf{y}\right).$$

*Proof.* According to our analysis in the lecture, we know

$$C(\mathbf{w}) = \frac{1}{2n}(X\mathbf{w} - \mathbf{y})^\top (X\mathbf{w} - \mathbf{y}) + \frac{\lambda}{2}\|\mathbf{w}\|_2^2.$$

One can show that $\nabla\|\mathbf{w}\|_2^2 = 2\mathbf{w}$ and therefore

$$\nabla C(\mathbf{w}) = \frac{1}{2n}\left(2X^\top X\mathbf{w} - 2X^\top \mathbf{y}\right) + \lambda\mathbf{w}.$$

According to the first-order optimality condition, we know $\nabla C(\mathbf{w}^*) = 0$, i.e.,

$$\nabla C(\mathbf{w}^*) = \frac{1}{n}\left(X^\top X\mathbf{w}^* - X^\top \mathbf{y}\right) + \lambda\mathbf{w}^* = 0 \Longrightarrow \left(\frac{1}{n}(X^\top X) + \lambda\mathbb{I}\right)\mathbf{w}^* = \frac{1}{n}X^\top \mathbf{y}.$$

It then follows that

$$\mathbf{w}^* = \left(\frac{1}{n}(X^\top X) + \lambda \mathbb{I}\right)^{-1}\left(\frac{1}{n}X^\top \mathbf{y}\right).$$

□

- A nice property is that we do not require to assume that the matrix $X^\top X$ is invertible. We can always find the above solution for the regularized least regression problem.
- If $\lambda = 0$, then this becomes the solution of the least squares regression problem. If $\lambda = \infty$, we get $\mathbf{w}^* = 0$, which is a trivial solution. We need to choose an appropriate $\lambda$.