AE 423 — Regression Analysis

Guodong Chen

Winter, 2019

1 Introduction

In statistical modeling, regression analysis is a set of statistical processes for estimating the relationships among variables [1]. More specifically, we are interested in how the value of the **output** (**target**, **response**) changes when the **inputs** (**features**, **states**) are varied. In a more statistical point of view, regression analysis estimates the conditional expectation of the outputs given the inputs.

Regression analysis is widely used in engineering for prediction and forecasting (sometimes called model or surrogate). For example, we can relate the lift coefficient of an airfoil c_l to the angle of attack α (thin airfoil theory, Figure 1), or force for stretching a spring F to the distance that the spring stretches Δx (Hooke's law). More interestingly, we can find a relation (build a model) between the the inputs and output of a more complex system. For example, instead of solving the Navier-Stokes equations every time which is extremely expensive, we can relate the aircraft shape (inputs) to the drag/lift (outputs) at cruise with a regression model (widely used in surrogate-based optimization, see Figure 2. The surrogate-based optimization is fast and cheap only for the optimization side, the surrogate building process is expensive).

Angle of attack, α(deg)	Lift coefficient, c_l
0.1904	0.0425
0.5511	0.0767
0.7387	0.0080
1.2137	0.1300
1.5541	0.1940
2.0318	0.1817
2.5218	0.2676
2.7390	0.3329
2.9900	0.3147
4.0685	0.3845
...	...

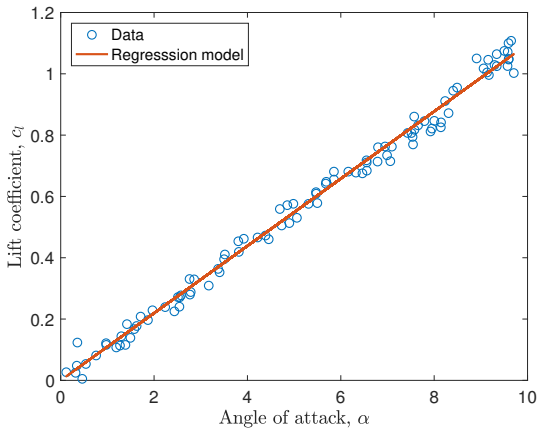

Figure 1: Lift coefficient c_l v.s. the angle of attack α of an airfoil

2 Regression Model

In regression analysis, we want to find the map f, parameterized by θ, from the inputs (**features**) $\mathcal{X} \in \mathbb{R}^n$ to a continuous output (**target**) $\mathcal{Y} \in \mathbb{R}$, $f : \mathcal{X} \mapsto \mathcal{Y}$, based on our observations or data (The notation and some explanations here follow Andrew Ng's lecture notes [2]). Often, the form of the function f is specified based on knowledge about the relationship between $\mathcal{Y}$ and X, or chosen to have good approximation power. However, the parameters $\theta \in \mathbb{R}^d$ in function f are unknown, which should be inferred from the data. For instance, we have a set of m realizations of $\mathcal{X}$ and $\mathcal{Y}$, $S = \{(x^i, y^i); i = 1, 2, ..., m\}$, which is often called **training set**. Each sample in the training set

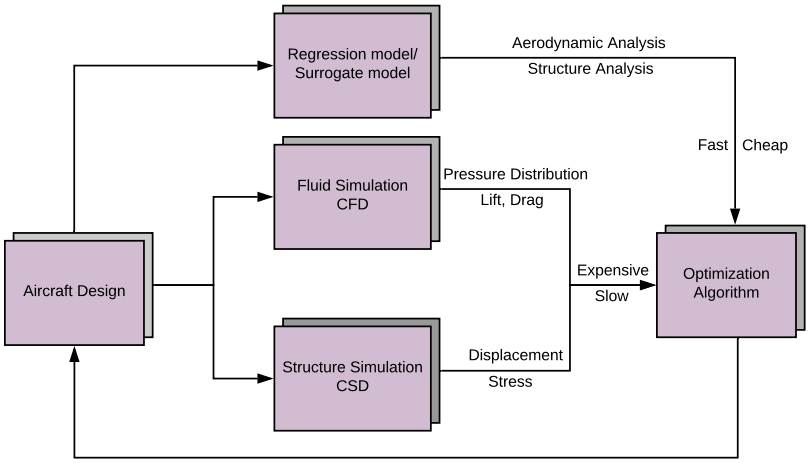

Figure 2: Regression model in aircraft optimization

has an input vector x^i of dimension n containing n independent features, y^i is the corresponding output (target), a scalar value. Then the regression process (or often called learning process by the machine learning community) can be stated as: find the optimal set of parameters θ^* such that the overall difference (error) between the model prediction $f(x^i;\theta^*)$ and the observation y^i is minimized. A diagram of the regression/learning process is shown as below.

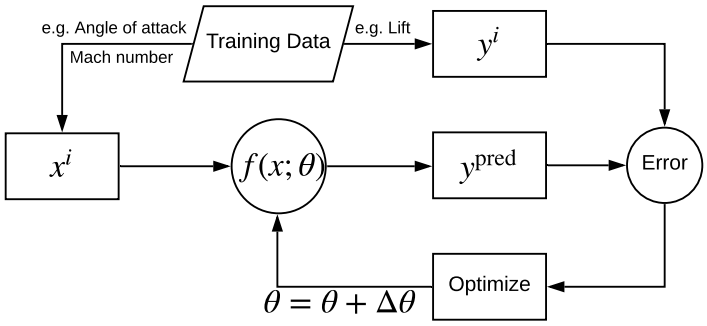

Figure 3: Regression/Learning process given a training data set

2.1 Linear Regression

We will start with the simplest type of the regression models: linear regression, in which the form of the regression function f is a linear function. Although linear regression is fairly simple, it's a very useful model in data analysis, and it can be easily extended to more complex models. With this setting, our linear regression model can be written as [1]

$$f(x;\theta) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ..., +\theta_n x_n, \tag{1}$$

where $\theta = [\theta_0, \theta_1, ..., \theta_n]^T$ are the parameters of our linear model. To simplify our notation, we also introduce the convention of letting x_0 (this is the **intercept term**), i.e., $x = [1, x]^T$, so that the

[1] Linear regression or linear model is not necessary a line or a hyperplane, e.g. if the features is $x = [x_1, x_1^2, x_2, x_3]^T$, then linear regression model is linear for x_2 and x_3, while it is quadratic for x_1. But if we consider x_1 and x_1^2 as two different features, the model is linear for these two features.

regression model can be written as

$$f(x;\theta) = \sum_{i=0}^{n} \theta_i x_i = \theta^T x, \tag{2}$$

here both θ and x are vectors of dimension n, where n is the dimension of the inputs. **Note:** x_j^i means the j^{th} component (feature) of i^{th} data in the training set.

Now, given a set of training data (observations), how can we pick, or lean the optimal parameters θ? One natural idea is to minimize the difference, often called the error, between the model prediction $f(x;\theta)$ and the observation y on the training data. The error measure can be any bounded vector norm, while L_1 and L_2 norms are the most common ones.

Regression in matrix form: For better exposition, we introduce the matrix form of the regression on the training data set. Given a training set, we can define the **design matrix** X to be the m by n matrix (m by $n+1$ if we include the intercept term) that contains the training inputs as its rows,

$$X = \begin{bmatrix} — & (x^1)^T & — \\ — & (x^2)^T & — \\ & \vdots & \\ — & (x^m)^T & — \end{bmatrix}. \tag{3}$$

Similarly, we can define the **output vector**, which contains all the target values from the training set,

$$Y = \begin{bmatrix} y^1 \\ y^2 \\ \vdots \\ y^m \end{bmatrix}. \tag{4}$$

With the definition above, we can easily write down the model prediction on the training data set as

$$\tilde{Y} = f(X;\theta) = X\theta, \tag{5}$$

thus the difference can be measured as

$$J = \|X\theta - Y\|_p. \tag{6}$$

If we choose the L_2 norm as the error measure, then the regression problem can be stated as an optimization problem [2],

$$\begin{aligned} \theta^* &= \arg\min_\theta \ \|X\theta - Y\|_2 \\ &= \arg\min_\theta \ \frac{1}{2}\|X\theta - Y\|_2^2 \\ &= \arg\min_\theta \ \frac{1}{2}(X\theta - Y)^T(X\theta - Y) \\ &= \arg\min_\theta \ \frac{1}{2}\sum_{i=1}^{m}(\theta^T x^i - y^i)^2 \\ &= \arg\min_\theta \ J(\theta). \end{aligned} \tag{7}$$

[2] $\arg\max$, defined as the arguments of the maxima, are the points in the domain of some function at which the function values are maximized, i.e., $\arg\max_\theta J(\theta)$ returns the maxima points θ^* of $J(\theta)$. Similarly, $\arg\min_\theta$ returns the minima points of the function

Here $J(\theta)$ is often called **objective function** or **loss function**, **cost function**. Since J takes the form of sum of squared error, this problem is also called **Ordinary Least Squares (OLS) regression**.

If instead we choose the L_1 norm as the error measure, we arrive at

$$\begin{aligned}\theta^* &= \arg\min_\theta \|X\theta - Y\|_1 \\ &= \arg\min_\theta \sum_i |\theta^T x^i - y^i| \\ &= \arg\min_\theta J(\theta),\end{aligned} \tag{8}$$

here, the cost function takes the form of the sum of absolute deviation of the prediction, thus it's called Least Absolute Deviations (LAD) regression. Ordinary Least Squares regression is more commonly used in practice, mainly because of its nicer math properties, e.g., differentiable, efficient vectorized computation. However, Ordinary Least Squares is not as robust as Least Absolute Deviations when there is noise or outliers (bad data, measurement error), see Section 2.4 for details.

Linear regression in a probabilistic point of view: Assume the targets and inputs are related via our regression model and an error term,

$$y^i = \theta^T x^i + \epsilon^i. \tag{9}$$

Let us further assume that the ϵ^i are distributed IID (independently and identically distributed) according to a Gaussian distribution (also called a Normal distribution) with mean zero and variance σ^2, i.e., $\epsilon \sim \mathcal{N}(0, \sigma^2)$. Then Equation 9 implies that y^i is also from a Gaussian distribution but with mean as $\theta^T x^i$ and variance as σ^2, i.e., $y^i \sim \mathcal{N}(\theta^T x^i, \sigma^2)$. We should note that this distribution is a conditional distribution given the observation of x^i, and the distribution is parametrized by θ. More precisely we should say $y^i | x^i; \theta \sim \mathcal{N}(\theta^T x^i, \sigma^2)$. Then given the observation of x^i, the probability of observing y^i as the output conditioned on the inputs can be written as

$$p(y^i | x^i; \theta) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^i - \theta^T x^i)^2}{2\sigma^2}\right). \tag{10}$$

Based on the independence assumption, the probability (**likelihood**) of observing outputs vector Y given the inputs X in the entire training set is

$$p(Y|X;\theta) = \prod_{i=1}^m p(y^i | x^i; \theta). \tag{11}$$

Note this quantity only depends on θ given a fixed training set, we often denote it as the **likelihood function** $L(\theta)$,

$$\begin{aligned}L(\theta) = p(Y|X;\theta) &= \prod_{i=1}^m p(y^i | x^i; \theta) \\ &= \prod_{i=1}^m \frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{(y^i - \theta^T x^i)^2}{2\sigma^2}\right).\end{aligned} \tag{12}$$

In probabilistic point of view, regression process is to find the most likely model conditional on what has been observed. Or conversely, the observed data on the best model should have the highest probability, so that we should choose the model parameters θ to maximize the likelihood function, which is called the **maximum likelihood estimation (MLE)**.

Since the exponential function is a little hard to maximize, we instead maximize the logarithm of the likelihood function, $\ell(\theta) = \log(L(\theta))$. These two maximization problems are equivalent since $\ell(\theta)$

is a strictly increase function of $L(\theta)$. Then the regression problem can be written as,

$$\begin{aligned}
\theta^* &= \arg\max_\theta \ \ell(\theta) \\
&= \arg\max_\theta \ \log(L(\theta)) \\
&= \arg\max_\theta \ \left[m \log(\frac{1}{\sqrt{2\pi}\sigma}) + \sum_{i=1}^{m} -\frac{(y^i - \theta^T x^i)^2}{2\sigma^2} \right] \\
&= \arg\max_\theta \ \sum_{i=1}^{m} -\frac{(y^i - \theta^T x^i)^2}{2\sigma^2} \\
&= \arg\min_\theta \ \frac{1}{2} \sum_{i=1}^{m} \frac{(y^i - \theta^T x^i)^2}{\sigma^2} \\
&= \arg\min_\theta \ \frac{1}{2} \sum_{i=1}^{m} (y^i - \theta^T x^i)^2 \quad \Longrightarrow \quad \text{OLS regression}
\end{aligned} \tag{13}$$

Here, we derive the OLS regression with a probabilistic point of view, by assuming the error of the regression model is in a Gaussian distribution. This assumption is not perfect but in general works pretty well in practice, since many error behaves like Gaussian distributed noise. Alternatively we can also assume the error obeys a Laplace distribution with zero mean and diversity b,

$$p(\epsilon) = \frac{1}{2b} \exp\left(-\frac{|\epsilon|}{b} \right) \tag{14}$$

then we can also derive the LAD regression with **MLE**.

2.2 Solving linear regression via optimization

As the regression problem is formulated in Section 2.1 as an optimization problem, it can be solved with any optimization algorithm. However, gradient-based optimization algorithm is preferred in practice since the gradient of the objective (loss) function is readily available. For example in the steepest descent algorithm, we update the parameters along the opposite direction of the objective gradient,

$$\theta = \theta - \alpha \nabla J(\theta), \tag{15}$$

where α is the update step length or often called **learning rate**, which can be determined by line search or can be specified by the user (use a constant or slowly decreasing rate).

Here we look at the OLS regression for instance, since it's gradient is analytical and easy to compute, that's also why it's more widely used in practice. And start even simpler, let's consider just one data pair from the training set, (x^i, y^i),

$$\begin{aligned}
\nabla J^i(\theta) &= \frac{\partial}{\partial \theta} \left(\frac{1}{2} (\theta^T x^i - y^i)^2 \right) \\
&= \underbrace{(\theta^T x^i - y^i)}_{\text{scalar}} \underbrace{x^i}_{\text{vector}} .
\end{aligned} \tag{16}$$

Then the gradient of the loss function is just the sum of the gradient caused by each data pair,

$$\begin{aligned}\nabla J(\theta) &= \frac{\partial}{\partial\theta}\left(\frac{1}{2}\sum_{i=1}^{m}(\theta^T x^i - y^i)^2\right)\\ &= \sum_{i=1}^{m}\frac{\partial}{\partial\theta}\left(\frac{1}{2}(\theta^T x^i - y^i)^2\right)\\ &= \sum_{i=1}^{m}(\theta^T x^i - y^i)x^i\\ &= \sum_{i=1}^{m}\nabla^i J(\theta).\end{aligned} \tag{17}$$

Just as before, we can also write this down in a matrix form,

$$\begin{aligned}\nabla J(\theta) &= \frac{\partial J(\theta)}{\partial\theta}\\ &= \frac{\partial}{\partial\theta}\frac{1}{2}(X\theta - Y)^T(X\theta - Y)\\ &= \frac{\partial}{\partial\theta}\frac{1}{2}(\theta^T X^T X\theta - 2\theta^T X^T Y + Y^T Y)\\ &= X^T X\theta - X^T Y.\end{aligned} \tag{18}$$

After getting the gradient of the loss function, we can simply get the steepest descent algorithm, also called **Batch Gradient Descent (BGD)** sometimes, as shown in Algorithm 1. In this algorithm, we

Algorithm 1 Batch Gradiecnt Descent

$\theta = \theta_0$ initialization
if not converge **then**
  $\nabla J(\theta) = \mathbf{0}$
  for i = 1:m **do**
    $\nabla J(\theta) = \nabla J(\theta) + (\theta^T x^i - y^i)x^i \quad \Longrightarrow \quad \nabla J(\theta) = X^T X\theta - X^T Y$
  end for
  $\theta = \theta - \alpha\nabla J(\theta)$
end if

loop over the entire data set to get the gradient of the loss function, which requires $\mathcal{O}(mn)$ operations. This can be very expensive and memory inefficient when we have a very big data set, i.e. when m is huge (when we talking about big data). Instead of updating the parameters after going through all the examples in the training set, we can learn on the fly by looking at several samples or even just one single data pair. These optimization approaches fall into the category of **stochastic gradient descent** algorithms, which are shown in Algorithm 2-3. Although the name sounds fancy, the actual implementation is fairly easy.

Instead of scanning through the entire training set and then make a single update step, stochastic gradient descent can start making progress right away with a single sample (operation $\mathcal{O}(n)$ per update) or a mini-batch of the samples (operations $\mathcal{O}(m_b n)$ per update), and continues to make progress with each example (mini-batch) it looks at. Therefore, stochastic gradient descent often gets "close" to the minimum much faster than batch gradient descent. However, it may never "converge" to the minimum, and the parameters θ will keep oscillating around the minimum of $J(\theta)$ as the gradient is just based on one example at each step and can be very noisy (mini-batch smooths the gradient), so as the optimization; but in practice most of the values near the minimum will be good enough. For example, think about two-dimensional regression ($n = 2$), in which we want to fit a straight line. Theoretically, we only need two data points to determine the slope and the intercept (in practice never

Algorithm 2 Stochastic Gradient Descent (Look at one sample at a time)

$\theta = \theta_0$ initialization
if not converge **then**
  Randomly shuffle examples in the training set $\implies$ where stochastic comes in
  for i = 1:m **do**
    $\nabla J(\theta) = (\theta^T x^i - y^i)x^i$
    $\theta = \theta - \alpha \nabla J(\theta)$
  end for
end if

Algorithm 3 Mini-Batch Stochastic Gradient Descent (Look at m_b samples at a time)

$\theta = \theta_0$ initialization
Pick batch size m_b, normally m is dividable by m_b
if not converge **then**
  Randomly shuffle examples in the training set $\implies$ where stochastic comes in
  $\nabla J(\theta) = \mathbf{0}$; $l = 1$
  while $l <= m$ **do**
    for $i = l : l + m_b - 1$ **do**
      $\nabla J(\theta) = \nabla J(\theta) + (\theta^T x^i - y^i)x^i$
    end for
    $\theta = \theta - \alpha \nabla J(\theta)$
    $l = l + m_b - 1$
  end while
end if

do that since the data is not perfect, noise), mini-batch of 2 is perfect for gradient calculations, only few iterations of stochastic gradient descent are required. In practice, stochastic gradient descent is often preferred over batch gradient descent on large scale problem or on large data set.

An example of linear regression on the thin-airfoil data Figure 1 is shown in Figure 4, BGD and SGD are both implemented, and the data is sampled from $y = 0x_0 + 2\pi x_1 + \epsilon$, where ϵ is a Gaussian noise with zero mean and some variance.

2.3 Solving Least Squares Regression via Normal Equation

The ultimate goal of the gradient descent algorithm is to drive the gradient of the loss function to zero, which is essentially a stationary point of the objective function, but is this a maxima or minima? and is the solution unique? These question can be answered by the **normal equation**, which can be obtained by setting the gradient of the loss function to be zero (this is what the optimization does ideally).

$$\nabla J(\theta) = X^T X\theta - X^T Y = 0 \quad \implies \quad X^T(X\theta - Y) = 0 \tag{19}$$

This equation is called normal equation because it means the difference between the model prediction and the data $X\theta - Y$ has to be normal to the span of the column space of X (or row space of X^T). If X has a complete column space (all the features x_j are linearly independent), then $X^T X$ is a **symmetric positive definite (SPD)** matrix (invertible), so there is an unique solution of Equation 19,

$$\theta^* = (X^T X)^{-1} X^T Y. \tag{20}$$

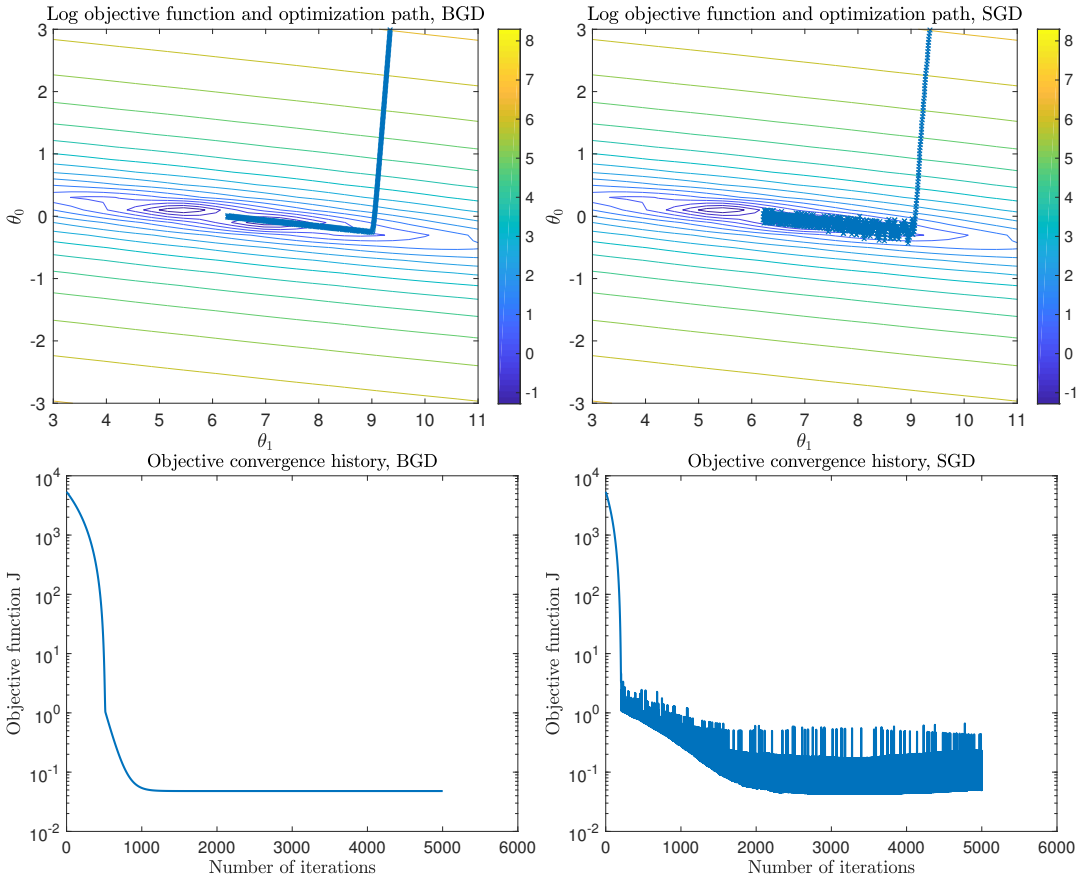

Figure 4: Comparison of BGD and SGD on a two-dimensional regression problem

But is this solution a minima or maxima? We can expand the loss function $J(\theta)$ at θ^* with a Taylor series,

$$\begin{aligned} J(\theta^* + \delta\theta) &= J(\theta^*) + \underbrace{(\nabla J(\theta^*))}_{=\mathbf{0}}{}^T \delta\theta + \frac{1}{2}\delta\theta^T \underbrace{\nabla^2 J(\theta^* + \beta\delta\theta)}_{\text{Hessian matrix, } X^TX} \delta\theta, \quad \beta \text{ is a scalar}, 0 \le \beta \le 1 \\ &= J(\theta^*) + \delta\theta^T X^T X \delta\theta \\ &= J(\theta^*) + (X\delta\theta)^T (X\delta\theta) \\ &= J(\theta^*) + \|X\delta\theta\|_2^2 \\ &\ge J(\theta^*) \qquad \text{if } X^TX \text{ is } \mathbf{SPD}\text{, strict inequality} \end{aligned} \tag{21}$$

Equation 21 means starting from the θ^*, no matter in which direction we perturb the parameters $\delta\theta$, we will increase the loss function. Therefore we can know the solution of the normal equation θ^* is a minima, and also because it's a unique solution, it's the global minima of our optimization problem.

Solving normal equation involves a matrix ($n \times n$, n is the number of features, not the number of data samples m). When we have a very large model or a very big data set, the matrix inversion can be very expensive, and X^TX can be very ill-conditioned. Thus, the optimization approach is more general in practice.

2.4 Robust Linear Regression

In the OLS regression, the sample data pairs that contribute most to the loss function and also the gradient are those which are furthest to the true model. What does this implies? Imagine we are using SGD algorithm to do the regression and assume we start at the true parameters (perfect initial guess), when we look at one sample exactly at the true model line, this sample does not produce any loss, so no gradient can be measured. However, if we are at a point which is far away from the true model line (noise in the data or measurement error), the loss and the gradient are very large which will in fact affect the parameters a lot in the gradient descent update. This can be worse if we have several completely wrong data in our training set. In these scenarios, how can we make the regression more robust? There are two options available.

- Locally weighted regression, the points away from the model are assigned with smaller weights. One possible weights can be

$$\begin{aligned} w^i &= \frac{1}{\sqrt{1 + (r^i)^2}} \\ r^i &= \theta^T x^i - y^i \end{aligned} \tag{22}$$

  Then the OLS can be written as

$$\theta^* = \arg\min_\theta \frac{1}{2} \sum_{i=1}^m w^i (y^i - \theta^T x^i)^2 \tag{23}$$

- Choose different loss function, OLS is not robust since the error gets squared, loss increases squarely as the distance from the model increases. LAD regression is more robust as the absolute deviation function only linearly depends on the deviation. More general, for any loss function, we can define our regression model as

$$\arg\min_\theta \frac{1}{2} \sum_{i=1}^m \rho(d^i), \qquad d^i = \theta^T x^i - y^i, \text{ is the model deviation} \tag{24}$$

  Several different loss function $\rho(d)$ are compared in Figure 5.

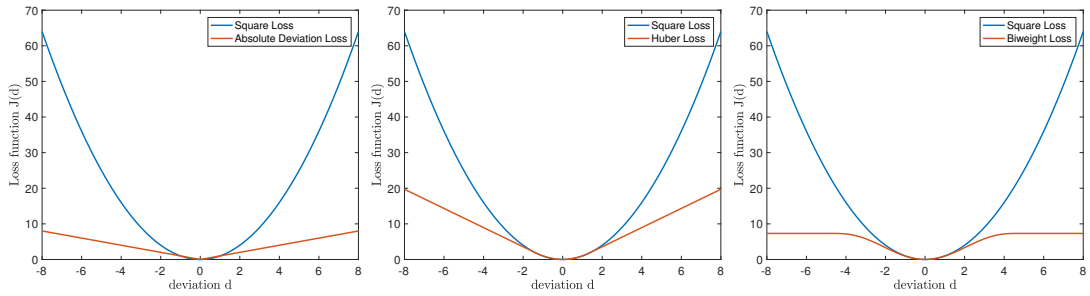

Figure 5: Several robust regression loss function compared to squared error

2.5 Under-fitting, Over-fitting and Regularization

Given the data, we can either solve the normal equation or the optimization problem to find the optimal parameters (build the model), but one may ask is the model good? One way is to test your model on unseen data, which is not used to train your model. If the model works well (predict accurate outputs) on the testing data set, most likely we've got a good model. But sometimes even before testing on the test data, we can have an idea of how good the model is by just looking at our model and data. Figure 6 gives an idea how the models may look like in practice, the data is still the airfoil data, but this time we have linear region and nonlinear region (stall, flow separation). The middle model is the best model here since it captures the main relations between the input and output while keep the model simple. The left model is a straight line (only two parameters, slope and intercept), which is too simple to be able to represent the relation, we call this situation under-fitting. For the model on the right, we see it can perfectly match all the training data, is this good? We can memorize all the data we have seen, but learn nothing from that. This case is often referred as over-fitting, which means the model is too complex (redundant parameters), paying too much on the training set while generalize badly (can't predict the output well) on the testing set. This is very common in practice, as we often tend to pick a complex model (don't know how many features may affect the output, often we choose as many candidate features as we can) when we do the regression.

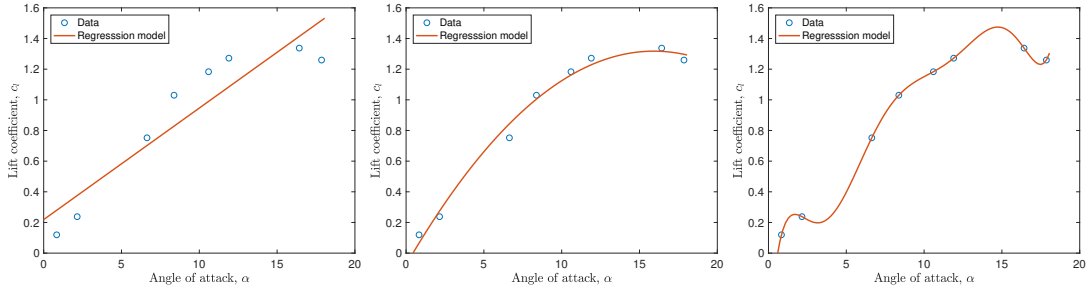

Figure 6: Three models, left one under-fitted the data, right one over-fitted the data

So how can we avoid over-fitting in practice, in a more automated way? We add regularization terms in the loss function.

$$J(\theta) = \frac{1}{2}\|X\theta - Y\|_2^2 + \lambda\|\theta\|_2^2 \tag{25}$$

where λ is the regularization factor, when $\lambda = 0$, we recover the OLS regression. The intuition is that, we do not want the parameters θ to have value in all its components, so we penalize it when big values appear in θ. In a probabilistic point of view, we are putting some constraints on the parameters θ, which can be from empirical knowledge or valid assumptions. When we estimate the parameters, we have a prior probability of θ, which may be the mean of θ is close to zero. Then instead of estimating the parameters solely based on the likelihood (data/observations), we also take our experience (prior probability) into account, this is called **maximum a posterior estimates**

(MAP). Similarly, we can put L_1 norm of θ as the regularization. The regularization with L_2 norm is also called **Ridge Regression**, and L_1 regularization is often called **Least Absolute Shrinkage and Selection Operator, LASSO**.

In practice, LASSO is often of more interest as the built-in sparsity in the model, which means by finding the model we also find the most important features that affect the output. Why does this happen? This is because L_1 norm prefers one large entry with all zeros to all small values. For example, if we consider two vector $\theta_1 = [4, 3]$ and $\theta_2 = [6, 0]$, we have $\|\theta_1\|_2 \leq \|\theta_2\|_2$ and $\|\theta_1\|_1 \geq \|\theta_2\|_1$. Application: discovering the governing equations (link to the paper), as shown in Figure 7.

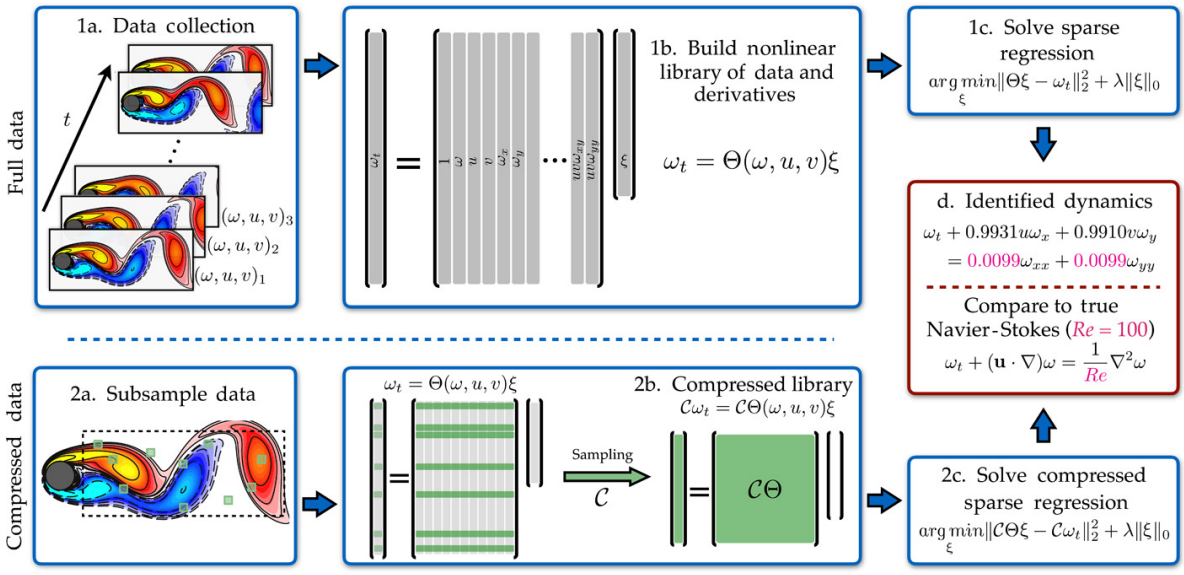

Figure 7: Using sparse regression (LASSO) to discover unknown governing equation, figure from [3]

3 Generalized Linear Regression

Back to a general regression problem, we want to find a map to best describe the relation between our input features and output target, $y = f(x; \theta)$, which is parametrized by a set of parameters θ. In linear regression, we assume the form of f as linear, which is not always the case. If we want a model with better approximation power than linear regression, it should be able to mimic nonlinear phenomenons. In this case, we need to specify a more complex form of $f(x; \theta)$. Build on linear model, we can compose it with some nonlinear function to get a more complicated (powerful) model. For example,

$$\begin{aligned} y &= g(h; \theta) = \theta^T h; \quad \dim(\theta) = l \times 1 \\ h &= \sigma(z) = \frac{1}{1 + \exp(-z)}; \quad \Longrightarrow \text{Nonlinear activation}, \dim(h) = \dim(z) = l \times 1 \\ z &= \ell(x; \Theta) = \Theta^T x; \quad \dim(\Theta) = l \times n, \dim(x) = n \times 1, \dim(z) = l \times 1 \end{aligned} \tag{26}$$

Then the final model can be written as,

$$y = g(\sigma(\ell(x)); \theta, \Theta) = f(x; \theta, \Theta) \tag{27}$$

By optimizing the parameters θ and Θ, we get a simple three-layer **neural network**! The structure is shown in Figure 8. The hidden layer structure in Figure 8 can be stacked to get deeper and deeper neural networks, which is widely used nowadays in regression and classification problems.

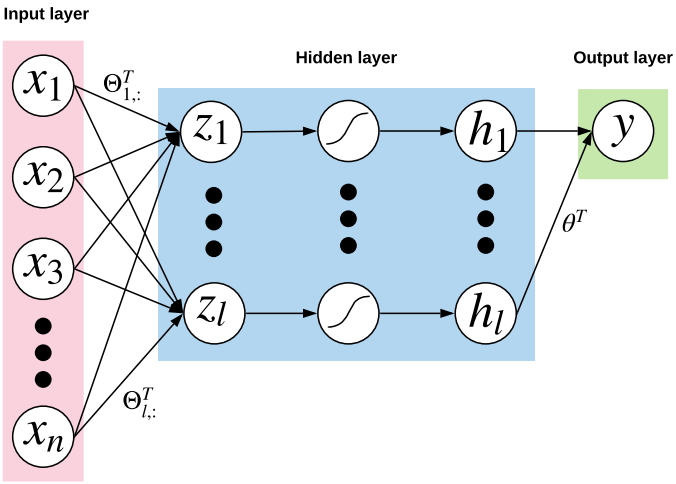

Figure 8: An example of simple neural networks

References

[1] Wikipedia. Regression analysis. https://en.wikipedia.org/wiki/Regression_analysis.

[2] Andrew Ng and John Duchi. Cs229: Machine learning. *Standford University Lecture*, 2000.

[3] Samuel H. Rudy, Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. Data-driven discovery of partial differential equations. *Science Advances*, 3(4), 2017.