Linear Regression with One Variable

Model and Cost Function

Model Representation

- **Supervised Learning (监督学习)**: Given the "right answer" for each example in the data.
  - **Regression Problem (回归问题)**: Predict real-valued output.
  - **Classification Problem (分类问题)**: Predict discrete-valued output.
- **Training set (训练集)**
  - **m**: number of training examples
  - **x**'s: "input" variable / features
  - **y**'s: "output" variable / "target" variable
  - (x, y): one training example
  - (x^i, y^i): i^{th} training example
- Training Set -> Learning Algorithm -> **h(hypothesis, 假设)**
  - h is a function maps from x's to y's
  - e.g. Size of house -> h -> Estimated price
- **Linear regression with one variable**
  - $h_\theta(x) = \theta_0 + \theta_1 x$
    - Shorthand: $h(x)$
  - Or named Univariate linear regression (单变量线性回归)

Cost Function

- **Hypothesis**: $h_\theta(x) = \theta_0 + \theta_1 x$
  - θ_i's: Parameters (模型参数)
  - How to choose θ_i's ?
    - Idea: Choose θ_0, θ_1 so that $h_\theta(x)$ is close to y for our training example (x, y)
- **Cost function (代价函数)**
  - $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$
  - Sometimes called Square error function (平方误差代价函数)
- Goal: minimise $J(\theta_0, \theta_1)$

Parameter Learning

Gradient Descent

- **Gradient Descent (梯度下降)**
  - Goal
    - Have some function $J(\theta_0, \theta_1)$
    - Want θ_0, θ_1 of $minJ(\theta_0, \theta_1)$
  - Outline
    - Start with some θ_0, θ_1, usually all set to 0.
    - Keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$ until we hopefully end up at minimum
- Gradient descent algorithm

  > repeat until convergence (收敛) {
  > $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_j)$ (for $j = 0$ and $j = 1$)
  > }

  - `:=` denotes assignment
  - α denotes learning rate
    - if too small, gradient descent can be slow
    - If too large, gradient descent can overshoot the minimum. It may fail to converge or even diverge.
  - You should <u>simultaneously</u> update θ_0 and θ_1
    - That is, you should compute the right-hand sides of θ_0 and θ_1, then save them to temporary variables, and finally update θ_0 and θ_1.

      > $temp0 := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_j)$
      >
      > $temp1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_j)$
      >
      > $\theta_0 := temp0$
      >
      > $\theta_1 := temp1$

Intuition

- If θ_1 at local optima, it leaves θ_1 unchanged.
- <u>gradient descent can converge to a local minimum, even with the learning rate α fixed.</u>
  - As we approach a local minimum, gradient descent will automatically take smaller steps. So, no need to decrease α over time.

Gradient Descent For Linear Regression

We can compute that

$\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)$

$\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x^{(i)}$

Thus the Gradient descent algorithm can be expressed as

> repeat until convergence {
> $\theta_0 := \theta_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)$
>
> $\theta_1 := \theta_1 - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right) \cdot x^{(i)}$
> }

And the cost funciton of linear refression is always a convex function (凸函数), or called Bowl-shaped function (弓形函数). <u>It doesn't have any local optima except for the one global optimum.</u>

"Batch" Gradient Descent

- The algorithm that we just went over is sometimes called **Batch Gradient Descent (批量梯度下降)**.
- "Batch": Each step of gradient descent uses all th etraining examples.