Lecture 1

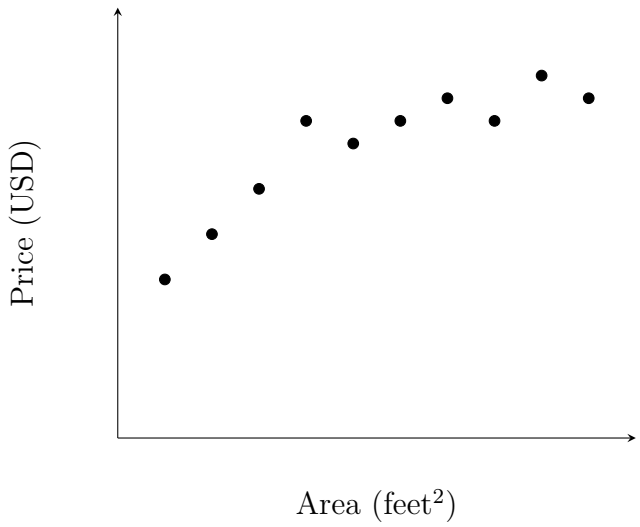
1. Basic notations
2. Linear regression
3. Gradient descent
4. Normal equations

1 Basic notations

- Arthur Samuel (1959). Machine Learning: field of study that gives computers the ability to learn without being explicitly programmed. He wrote checker program computer against himself. Computer learned how to play checkers better than Arthur Samuel.
- Tom Mitchell (1998) Well-posed learning problem: A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.

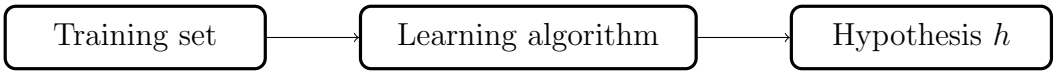
Let us consider the example: we will try to predict housing prices. Features: living area (feet^2); output: price. The question is to find the relationship between living area and price.

Living area (feet^2)	**USD** ($\times$1000)
2104	400
1416	232
1534	315
852	178
1940	240
...	...



Notations:

- m: number of training examples
- x: input variables or features (notice that x is a column vector)
- y: output variable or target
- (x, y): training example (sample)
- $(x^{(i)}, y^{(i)})$: i-th training example



2 Linear regression

We will choose the hypothesis in the following form:

$$h(x) = \theta_0 + \theta_1 x$$

More complicated example: we add number of bedrooms in the houses, x_1 is a size, x_2 is a number of bedrooms, then the hypothesis:

$$h(x) = h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

For conciseness, define $x_0 = 1$, then

$$h(x) = \sum_{i=0}^{n} \theta_i x_i = \theta^T x,$$

where n is a number of features, θ_i are **parameters**. The task of learning algorithm is to learn parameters from the training set. To learn the parameters we should choose the cost (error) function. The natural choice is

$$J(\theta) = \frac{1}{2} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

and we try to find

$$\min_\theta \frac{1}{2} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2.$$

3 Gradient descent

The idea of gradient descent is to start from the initial random value of θ (say $\theta = \vec{0}$). Then we keep changing θ to reduce $J(\theta)$. The new point could be obtained by choosing the direction of steepest descent (because the aim is to go down as quickly as possible). The procedure converges to the local minimum of $J(\theta)$. The problem is that if we start from different initial point, then we could find another minimum. If the function $J(\theta)$ is very complicated, then it could have a lot of minima (Fig. 1).

In more details: we update θ on each iteration by the following rule

$$\theta_i := \theta_i - \alpha \frac{\partial}{\partial \theta_i} J(\theta)$$

(notice, this is an operation of assignment).

We will find the partial derivative for one training example:

$$\frac{\partial}{\partial \theta_i} J(\theta) = \frac{\partial}{\partial \theta_i} \left(\frac{1}{2} (h_\theta(x) - y)^2 \right) = 2 \cdot \frac{1}{2} \cdot (h_\theta(x) - y) \cdot \frac{\partial}{\partial \theta_i} (h_\theta(x) - y) =$$
$$= (h_\theta(x) - y) \cdot \frac{\partial}{\partial \theta_i} (\theta_0 x_0 + \ldots + \theta_n x_n - y) = (h_\theta(x) - y) \cdot x_i$$

Then the update will be

$$\theta_i := \theta_i - \alpha (h_\theta(x) - y) \cdot x_i,$$

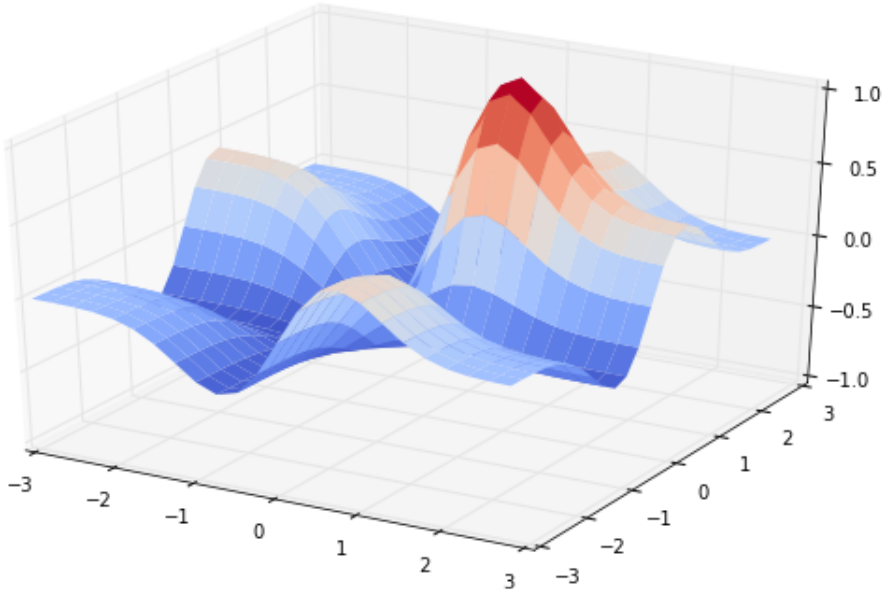


Figure 1: Function with many minima

where α is a **learning rate**, which shows how large your step in the gradient descent. If α is too small, the algorithm takes long time to converge, if α is too big, the algorithm can be diverge.

Then the algorithm becomes very simple.

Algorithm 1 Batch Gradient Descent

1: **repeat**
2: **for** $i = 0$ **to** n **do**
3: $\theta_i := \theta_i - \alpha \sum_{j=1}^{m} (h_\theta(x^{(j)}) - y^{(j)}) \cdot x_i^{(j)}$
4: **until** convergence

Notice again that the second part in the line 3 of the Algorithm 1 is just $\frac{\partial}{\partial \theta_i} J(\theta)$.

In our problem, $J(\theta)$ does not have a complicated form, it is just quadratic surface. It converges reasonably rapidly. The gradient is decreasing (the step becomes smaller and smaller).

The name of the algorithm Batch Gradient Descent came from the idea that we look at the whole training set every step. But when training set is very big (for example, $m > 1000000$), then we need to look at a huge amount of training samples each iteration. The alternative for this is called Stochastic Gradient Descent.

Algorithm 2 Stochastic Gradient Descent

repeat
  for $j = 1$ **to** m **do**
    for $i = 0$ **to** n **do**
      $\theta_i := \theta_i - \alpha(h_\theta(x^{(j)}) - y^{(j)}) \cdot x_i^{(j)}$
until convergence

In this algorithm we update weights based on the first training example only, after on the second training example only and so on. For large datasets the Stochastic Gradient Descent is much faster. But the problem that you do not walk to the minimum in the fastest way but you still approaching to it (Fig. 2).

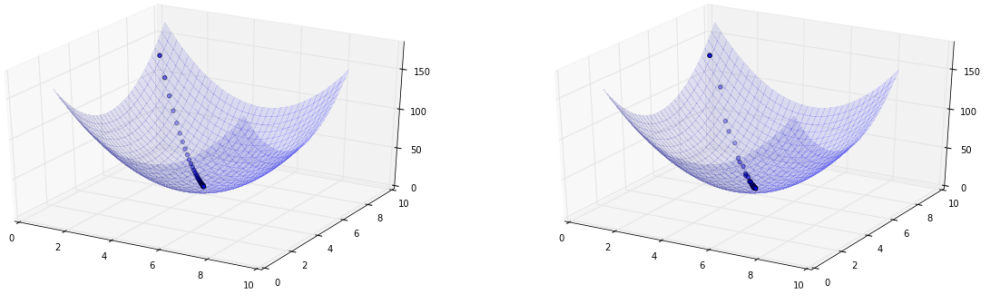


Figure 2: Batch vs Stochastic Gradient Descent

4 Normal equations

There is another way to perform minimization. We will derive the solution using Linear Algebra terminology. But we need to take derivatives with

respect to matrices. Given the function we need to find

$$\nabla_\theta J = \begin{bmatrix} \frac{\partial J}{\partial \theta_0} \\ \vdots \\ \frac{\partial J}{\partial \theta_n} \end{bmatrix} \in \mathbb{R}^{n+1}$$

Gradient descent iteration transforms to

$$\theta := \theta - \alpha \nabla_\theta J,$$

where left-hand and right-hand sides are $n+1$-dimensional vectors.

Definition. $f : \mathbb{R}^{m \times n} \to \mathbb{R}$ or $f(A), A \in \mathbb{R}^{m \times n}$. Then

$$\nabla_A f(A) = \begin{bmatrix} \frac{\partial f}{\partial A_{11}} & \cdots & \frac{\partial f}{\partial A_{1n}} \\ \vdots & & \vdots \\ \frac{\partial f}{\partial A_{m1}} & \cdots & \frac{\partial f}{\partial A_{mn}} \end{bmatrix}$$

Some facts from the Linear Algebra:

- $\operatorname{tr} AB = \operatorname{tr} BA$
- $\operatorname{tr} ABC = \operatorname{tr} CAB = \operatorname{tr} BCA$
- If $f(A) = \operatorname{tr} AB$, then $\nabla_A \operatorname{tr} AB = B^T$
- $\operatorname{tr} A = \operatorname{tr} A^T$
- If $a \in \mathbb{R}$ then $\operatorname{tr} a = a$
- $\nabla_A \operatorname{tr} ABA^T C = CAB + C^T AB^T$

Denote

$$X\theta = \begin{bmatrix} - - - (x^{(1)})^T - - - \\ - - - (x^{(2)})^T - - - \\ \cdots \\ - - - (x^{(m)})^T - - - \end{bmatrix} \theta = \begin{bmatrix} (x^{(1)})^T \theta \\ (x^{(2)})^T \theta \\ \cdots \\ (x^{(m)})^T \theta \end{bmatrix} = \begin{bmatrix} h_\theta(x^{(1)}) \\ h_\theta(x^{(2)}) \\ \cdots \\ h_\theta(x^{(m)}) \end{bmatrix}$$

Remember that

$$y = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix},$$

then

$$X\theta - y = \begin{bmatrix} h_\theta(x^{(1)}) - y^{(1)} \\ h_\theta(x^{(2)}) - y^{(2)} \\ \cdots \\ h_\theta(x^{(m)}) - y^{(m)} \end{bmatrix}.$$

Recall $z^T z = \sum_i z_i^2$. Then

$$\frac{1}{2}(X\theta - \vec{y})^T(X\theta - \vec{y}) = \frac{1}{2}\sum_{i=1}^{m}(h(x^{(i)}) - y^{(i)})^2 = J(\theta)$$

Now we should set gradient to zero:

$$\nabla_\theta J(\theta) = \vec{0}$$

It means

$$\nabla_\theta \left(\frac{1}{2}(X\theta - y)^T(X\theta - y)\right) = \frac{1}{2}\nabla_\theta \operatorname{tr}\left(\theta^T X^T X\theta - \theta^T X^T y - y^T X\theta + y^T y\right) =$$
$$= \frac{1}{2}\left(\nabla_\theta \operatorname{tr}\theta\theta^T X^T X - \nabla_\theta \operatorname{tr} y^T X\theta\right).$$

But

$$\nabla_\theta \operatorname{tr}\theta I\theta^T X^T X = X^T X\theta I + X^T X\theta I$$

and

$$\nabla_\theta \operatorname{tr} y^T X\theta = X^T y,$$

then

$$\nabla_\theta J(\theta) = \frac{1}{2}\left[X^T X\theta + X^T X\theta - X^T y - X^T y\right] = X^T X\theta - X^T y = 0.$$

The right hand side is called **normal equations**:

$$X^T X\theta = X^T y$$

and finally

$$\theta = (X^T X)^{-1} X^T y.$$