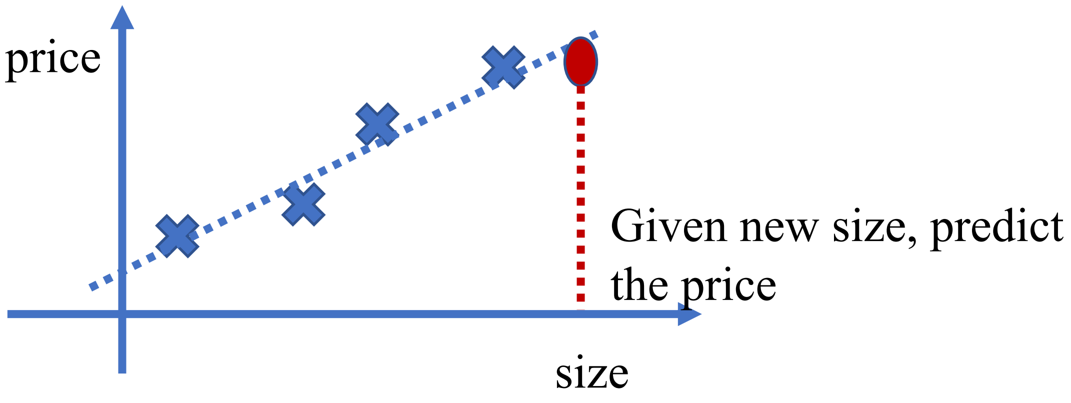
Linear Regression

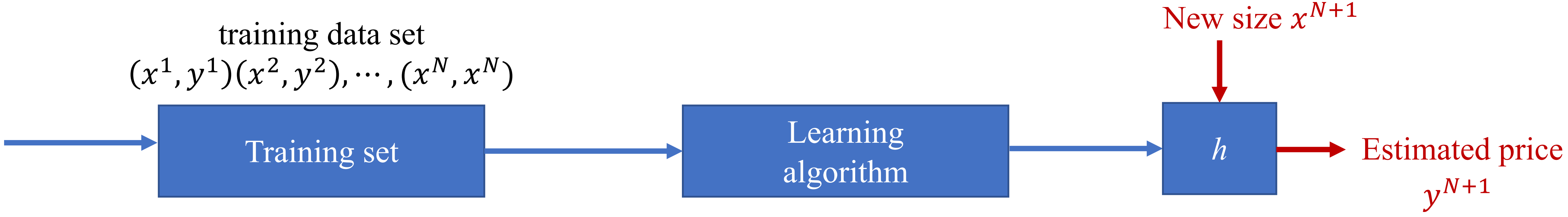
CIS492 Machine Learning

Dr. Qin Lin

House price prediction



Size (feet²)	Price ($1,000)
2104	400
1600	330
2400	369
1416	232
3000	540
…	…



In this example, $\mathcal{X} \in \mathbb{R}, \mathcal{Y} \in \mathbb{R}$

How to represent h?

In 1-d linear case, $h(x) = \theta_0 + \theta_1 x$

intercept slope

What if we take more features into consideration?

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

compact representation: $h(x) = \sum_{i=0}^{2} \theta_i x_i = \theta^T x$

$$\theta = \begin{pmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{pmatrix} \qquad x = \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ x_1 \\ x_2 \end{pmatrix}$$

Recap: $(\theta_0 \quad \theta_0 \quad \theta_0)\begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix}$, let x_0=1 to make inner product legal

For a general case, $h(x) = \sum_{i=0}^{d} \theta_i x_i = \theta^T x$
d: # of input variables (not counting x_0), or input/feature dimension

Size (feet^2)	#bedrooms	Price ($1,000)
2104	3	400
1600	3	330
2400	3	369
1416	2	232
3000	4	540
…	…	…

Learning algorithm's task is to choose θ, such that

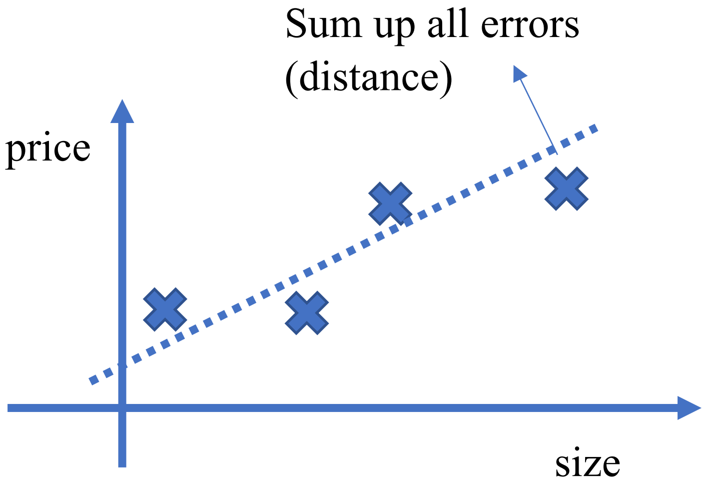
$h(x) \approx y$ at least for training examples

Define cost/loss function: $J(\theta)$

$$\min_{\theta} \frac{1}{2} \sum_{i=1}^{n} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$

For an easier computation, no influence on θ optimization

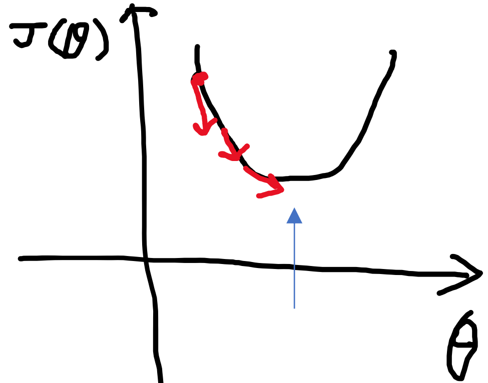
Why choose 2-norm?
Euclidian distance, can also use 1-norm ||, but 2-norm is most common, will have another explanation in "probabilistic interpretation" part



Always remember: it's a function of θ, after plunging in all known data

Gradient decent (also known as steepest descent)

Basic idea: to take repeated steps in the opposite direction of the gradient of the function at the current point, because this is the direction of the steepest descent

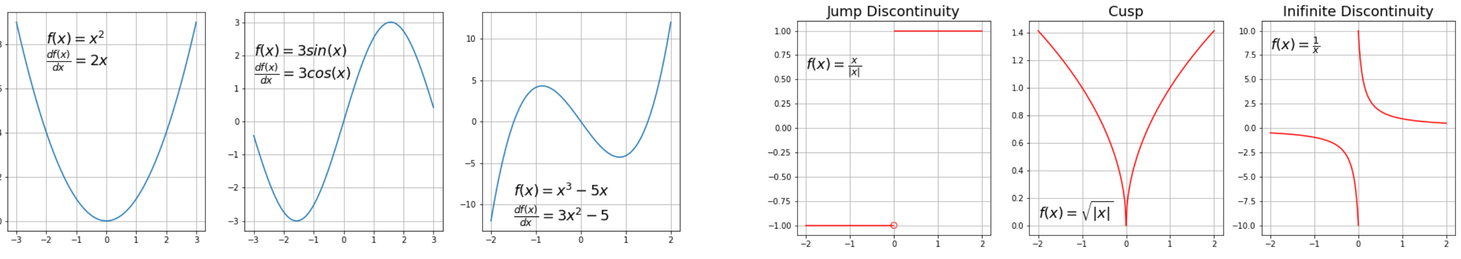


Want to find this minima

For a convex function, only one minima

Two conditions to use GD:

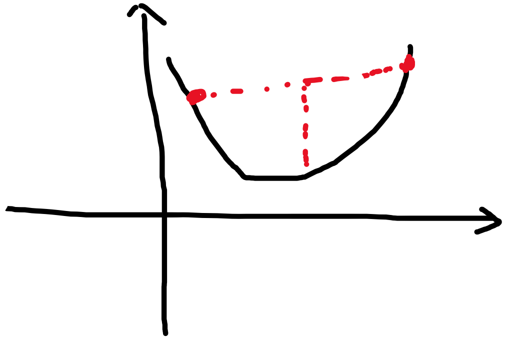
1) Differentiable

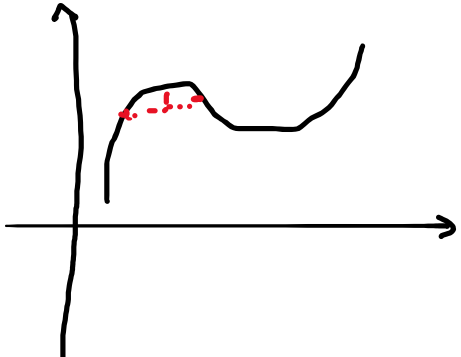


2) Convex

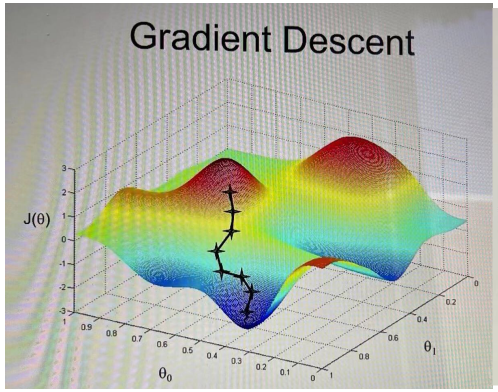
$$f(\lambda x_1 + (1-\lambda)x_2) \le \lambda f(x_1) + (1-\lambda) f(x_2)$$

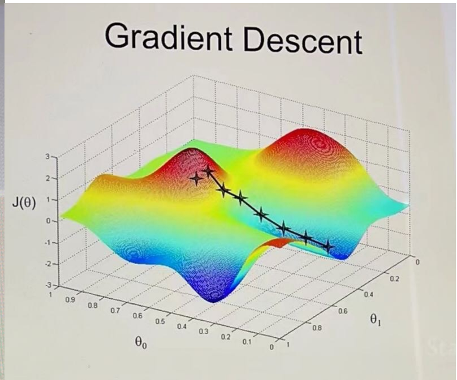
take $\lambda = 0.5$ as example





- For concave function $f(\lambda x_1 + (1-\lambda)x_2) \ge \lambda f(x_1) + (1-\lambda) f(x_2)$, we will need gradient ascent (will be covered in maximum likelihood estimation)
- For a general function, GD can only find local optima (see next slide)





For a general non-convex function, starting with different initial condition will get different results (one solution would be multiple runs with different initial conditions and compare)

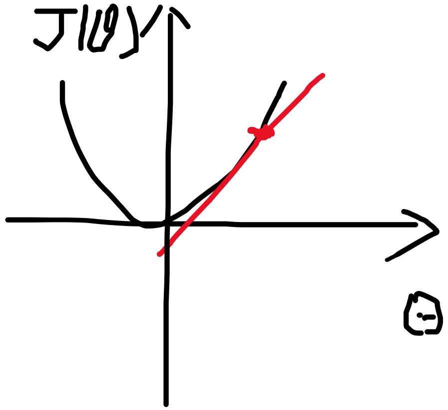
Gradient decent:

Partial derivative

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

Learning rate, step size to update

Why negative not positive sign?



$J(\theta) = \theta^2$

initial value $\theta = 2$

$$\frac{\partial J(\theta)}{\partial \theta} = 2\theta = 4$$

$\theta := \theta - \alpha * 4$

θ should be reduced

- Update rule for each dimension (break down the gradient direction into direction in each dimension)
- := "colon equal" means assignment, "*a*=*b*" is for statement, so *a*=*a*+*1* is wrong, should be *a*:=*a*+1
- Vector representation (update direction in multi-dimension)

$$\theta := \theta - \alpha \nabla J(\theta)$$

$$\nabla J(\theta) = \begin{pmatrix} \frac{\partial}{\partial \theta_0} J(\theta) \\ \frac{\partial}{\partial 1} J(\theta) \\ \vdots \\ \frac{\partial}{\partial \theta_d} J(\theta) \end{pmatrix}$$

$$\frac{\partial}{\partial\theta_j}J(\theta) = \frac{\partial}{\partial\theta_j}\frac{1}{2}(h_\theta(x) - y)^2$$

$$= 2 * \frac{1}{2}(h_\theta(x) - y)\frac{\partial}{\partial\theta_j}(h_\theta(x) - y)$$

$$= (h_\theta(x) - y)\frac{\partial}{\partial\theta_j}(\theta_0 x_0 + \theta_1 x_1 + \cdots + \theta_d x_d - y)$$

$$= (h_\theta(x) - y)x_j$$

e.g., if $j = 1, \frac{\partial}{\partial\theta_j}\theta_0 x_0, \frac{\partial}{\partial\theta_j}\theta_2 x_2, \cdots$ will all be zeros

Recap: chain rule
for $h(x) = f(g(x))$
$h'(x) = f'(g(x))g'(x)$

Update rule will be $\theta_j := \theta_j - \alpha(h_\theta(x) - y)x_j$ LMS (least mean square) update rule

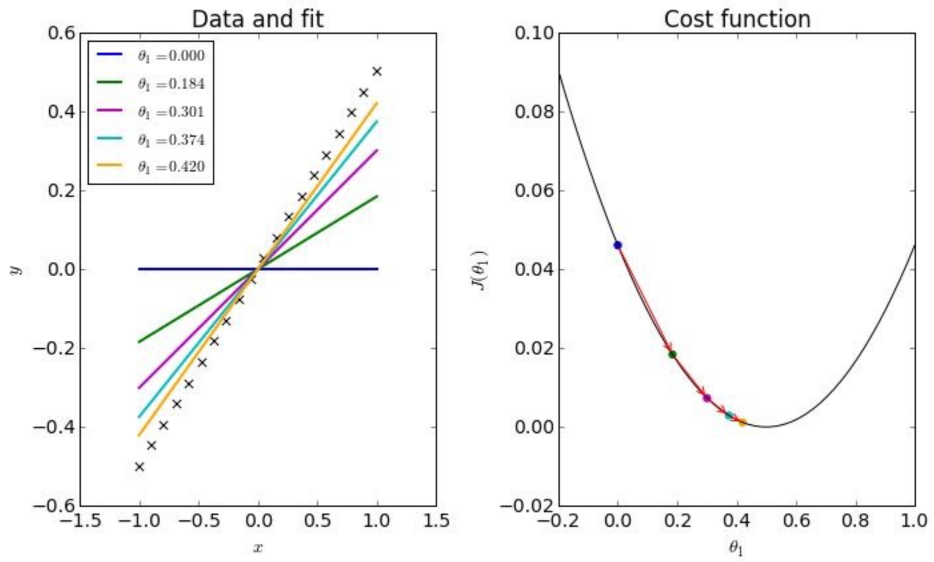
If $h_\theta(x)$ close to y, no need too much update with θ
otherwise, large update

What if we have multiple examples

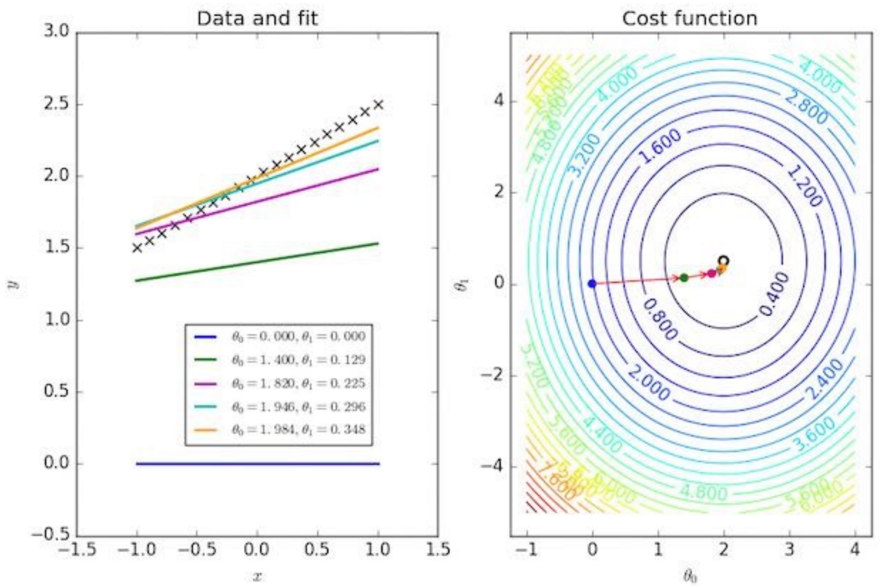
$$\theta_j := \theta_j - \alpha\frac{\partial}{\partial\theta_j}\sum_{i=1}^{n}\left(h^{(i)}(x) - y^{(i)}\right)^2$$

$$= \theta_j - \alpha\sum_{i=1}^{n}\left(h^{(i)}(x) - y^{(i)}\right)x_j^{(i)} \quad \text{for } j = 0,1,2,\cdots,d$$

Sum's derivative is equal to derivatives' sum



for 1-d optimization, we only optimize the slope, the cost function is just quadratic



for 2-d optimization, we optimize the slope and the intercept, the cost function is contour

Question: why the update direction is perpendicular to the tangent of contour

- Batch gradient decent (BGD)

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} \sum_{i=1}^{n} \frac{1}{2} \left(h^{(i)}(x) - y^{(i)}\right)^2$$

$$= \theta_j - \alpha \sum_{i=1}^{n} \left(h^{(i)}(x) - y^{(i)}\right) x_j^{(i)} \quad \text{for } j = 0,1,2,\cdots,d$$

- Stochastic gradient decent (SGD)

for i=1 to n {

$\theta_j := \theta_j - \alpha\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) x_j^{(i)},$ for all j

}

BGD

Pros: robust to noise, easy to converge

Cons: sum operation for big data is slow

SGD

Pros: fast

Cons: sensitive to noise, not easy to converge

Normal equation

Explicit method to minimize $J(\theta)$ instead of an iterative algorithm

Recall $J(\theta) = \frac{1}{2}\sum_{i=1}^{n}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$

1*(d+1)

$$X\theta = \begin{pmatrix} (x^{(1)})^T \\ (x^{(2)})^T \\ \vdots \\ (x^{(n)})^T \end{pmatrix} \theta = \begin{pmatrix} (x^{(1)})^T\theta \\ (x^{(2)})^T\theta \\ \vdots \\ (x^{(n)})^T\theta \end{pmatrix}$$

(d+1)*1

n*(d+1)

d: dimension of features

Extended to include intercept

e.g., x $= \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, d = 2, \theta = \begin{pmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{pmatrix}$ $x^T\theta = (1 \quad x_1 \quad x_2)\begin{pmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{pmatrix} = \theta_0 + \theta_1 x_1 + \theta_2 x_2$

$$X\theta - y = \begin{pmatrix} \left(x^{(1)}\right)^T \theta - y^{(1)} \\ \left(x^{(2)}\right)^T \theta - y^{(2)} \\ \vdots \\ \left(x^{(n)}\right)^T \theta - y^{(n)} \end{pmatrix}$$

$$for\ a\ vector\ z, z^T z = \sum_i z_i^2, sum\ of\ each\ element$$

$$\frac{1}{2}(X\theta - y)^T (X\theta - y) = \frac{1}{2}\sum_{i=1}^{n}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2 = J(\theta)$$

$$\nabla_\theta J(\theta) = \nabla_\theta \frac{1}{2}(X\theta - y)^T(X\theta - y)$$

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

$$= \frac{1}{2}\nabla_\theta[\theta^T(X^T X)\theta - y^T(X\theta) - y^T(X\theta)]$$

$$\because X \in \mathcal{R}^{n*(d+1)}, \qquad \theta \in \mathcal{R}^{(d+1)*1}$$

$$X\theta \in \mathcal{R}^{n*1}, y \in \mathcal{R}^{n*1},\ (X\theta)^T y = y^T(X\theta) \in \mathcal{R}, \text{scalar}$$

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

$$= \frac{1}{2}(2X^T X\theta - 2X^T y) = X^T X\theta - X^T y$$

$$let\ \nabla_\theta J(\theta) = 0$$

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

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

Linear algebra review

- $\nabla_x b^T x = b$
- $\nabla_x x^T A x = 2Ax$

c.f. Section 4.3, Linear Algebra Review and Reference

Q: normal equation and GD have (almost) same results in convex optimization (only one minima), which one is better?

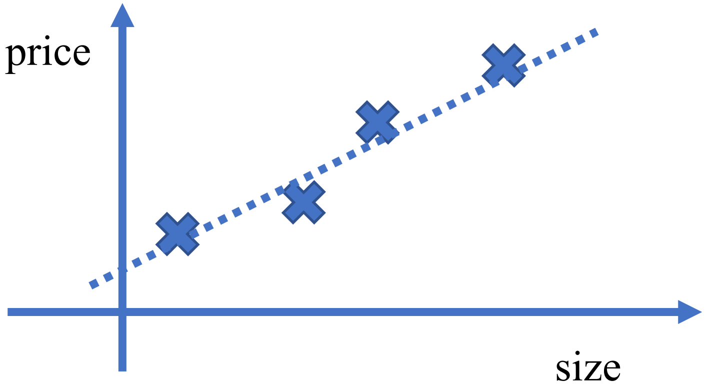
Normal equation needs matrix inverse (in general the complexity is $O(n^3)$), not suitable for big data

Locally weighted regression (LWR)

- Parametric learning algorithm

Fit fixed set of parameters θ for data

e.g., linear regression (once the training is done, in memory, just θ)

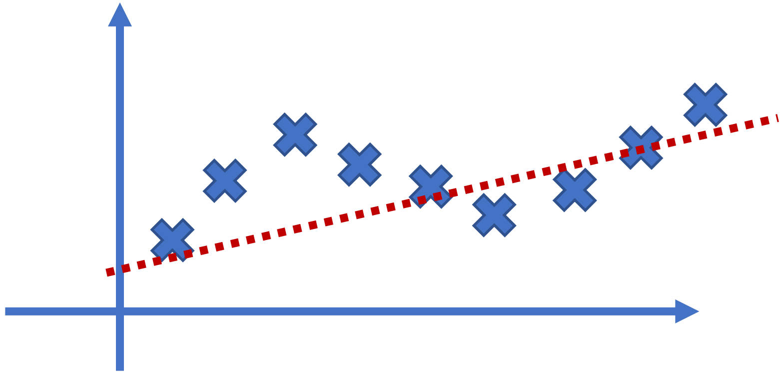


Steps for linear regression

1) $\min_{\theta} \frac{1}{2}\sum_{i=1}^{n}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$ for training data, return θ

2) $\theta^T x$ is used to predict

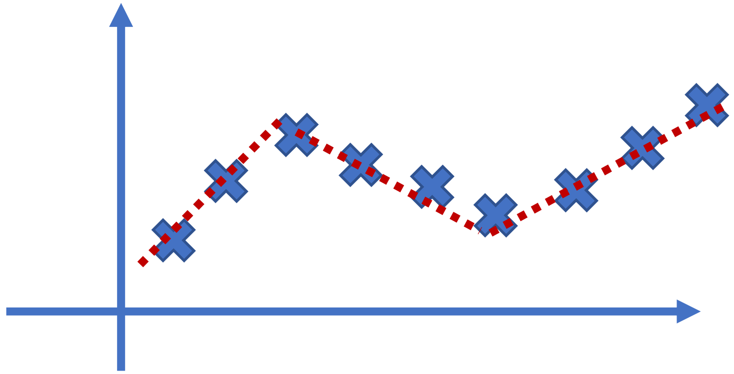
- Advantage: efficient in memory, once we get θ, we don't need the training data anymore (could be million)
- Disadvantage: only works well for linear



Linear regression could be very wrong

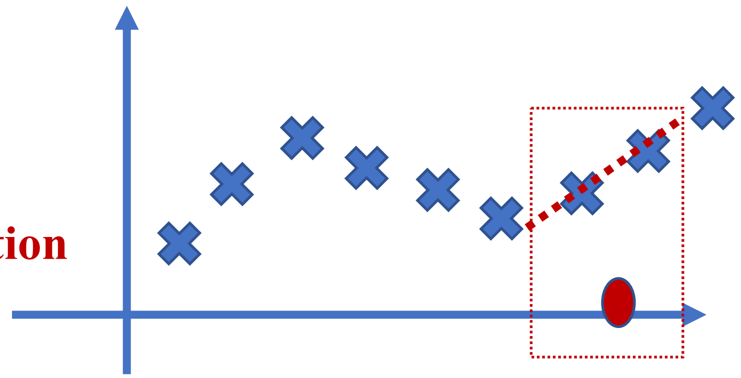
• Motivation of LWR

Only assume that some regions are linear, which is reasonable because a complex function can be piecewise linear



Basic idea of LWR

- Need to store all training data
- To predict given a datum x
- Its neighbor is more important than other data
- Fit a linear line (LMS or normal equation)



To predict, every time we need to run an optimization

- **Advantage: can learn very complex function**
- **Disadvantage: needs to store all training data (could be very large)**

The mechanism to "care" more about the neighbor is implemented by weighting

$$\min_{\theta} \frac{1}{2} \sum_{i=1}^{n} w^{(i)} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$

where $w^{(i)}$ is a weighted function, a common choice

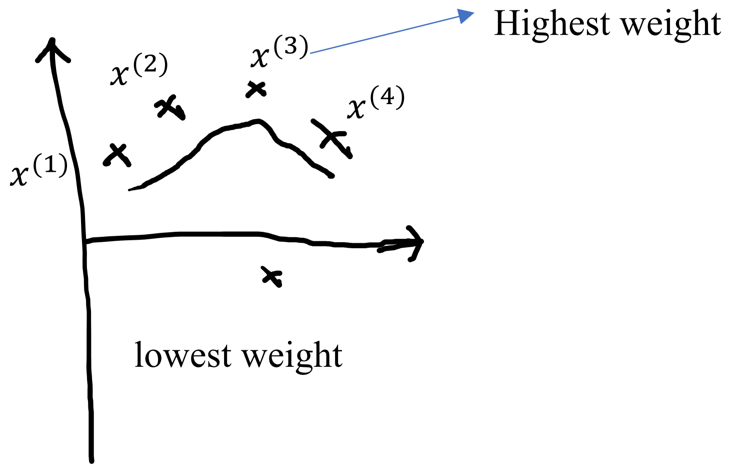
$$w^{(i)} = \exp\left(-\frac{\left(x^{(i)} - x\right)^2}{2\tau^2}\right)$$

query datum

bandwidth

Looks like Gaussian, but not exactly same, since the pdf sum is not 1

If $\left|x^{(i)} - x\right|$ is small, $w^{(i)} \approx 1$
If $\left|x^{(i)} - x\right|$ is large, $w^{(i)} \approx 0$
$w^{(i)}$: how much attention you need to pay for $\left(x^{(i)}, y^{(i)}\right)$
τ: bandwidth (shape of the function)



Query point $x = 5.0$

$$\begin{cases} x^{(1)} = 4.9, y^{(1)} \\ x^{(2)} = 3.0, y^{(2)} \end{cases}$$

Training data stored

$$w^{(i)} = exp\left(\frac{-\left(x^{(i)} - x\right)^2}{2\tau^2}\right), \tau = 0.5$$

$$w^{(i)} = exp\left(\frac{-(4.9 - 5.0)^2}{2 * (0.5)^2}\right) = 0.9802$$

$$w^{(i)} = exp\left(\frac{-(3.0 - 5.0)^2}{2 * (0.5)^2}\right) = 0.000335$$

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{n} w^{(i)}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2 = 0.5\left(0.9802 * \left(\theta^T x^{(1)} - y^{(1)}\right)^2 + 0.00035 * \left(\theta^T x^{(2)} - y^{(2)}\right)^2\right)$$

We pay more attention to $\{x^{(1)}, y^{(1)}\}$

so, every time for a new query (testing datum), we need to compute $w^{(i)}$ for each training datum, because it depends on x, then run optimization to solve and get θ

Probabilistic interpretation (why quadratic form $\min_{\theta} \frac{1}{2}\sum_{i=1}^{n}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$)

Assume the underlying model $y^{(i)} = \theta^T x^{(i)} + \varepsilon^{(i)}$ error, unmodeled effect

random noise $\varepsilon^{(i)} \sim N(0, \sigma^2)$

Probabilistic density distribution (pdf)

$$f\left(\varepsilon^{(i)}\right) = \frac{1}{\sqrt{2\pi}\sigma} exp\left(\frac{-\left(\varepsilon^{(i)}\right)^2}{2\sigma^2}\right)$$

i. i. d. (Independent and identically distributed) noise for different examples

This implies

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

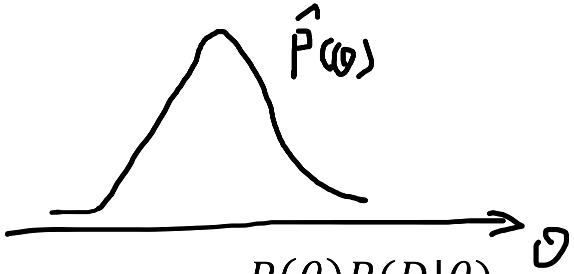
$i.e., y^{(i)} | x^{(i)}; \theta \sim N\left(\theta^T x^{(i)}, \sigma^2\right)$

We don't use $f\left(y^{(i)} | x^{(i)}, \theta\right)$ because it means that θ is also a random variable with distribution, but here we just consider θ as a fixed unknown variable

Frequentist and Bayesian debate

$\hat{\theta} = \arg\max_{\theta} P(D|\theta)$

maximum likelihood estimation



$$\hat{P}(\theta|D) = \frac{P(\theta)P(D|\theta)}{P(D)}$$

Bayesian infererence

Maximize likelihood $L(\theta)$, finding the optimal θ, such that the prob. of observing giving training data is maximized

$$\mathcal{L}(\theta) = f(y|x;\theta) = \prod_{i=1}^{n} f\left(y^{(i)}|x^{(i)};\theta\right)$$

$$= \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{-\left(y^{(i)} - \theta^T x^{(i)}\right)^2}{2\sigma^2}\right)$$

define a log−likelihood, just for easier computation, if likelihood is maximized, log−likelihood will be maximized too

$$L(\theta) = \log \mathcal{L}(\theta) = \log \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{-\left(y^{(i)} - \theta^T x^{(i)}\right)^2}{2\sigma^2}\right)$$

$$= \sum_{i=1}^{n} \left[\log \frac{1}{\sqrt{2\pi}\sigma} + \log \exp\left(\frac{-\left(y^{(i)} - \theta^T x^{(i)}\right)^2}{2\sigma^2}\right)\right]$$

$$= n \log \frac{1}{\sqrt{2\pi}\sigma} + \sum_{i=1}^{n} \frac{-\left(y^{(i)} - \theta^T x^{(i)}\right)^2}{2\sigma^2}$$

$$= n \log \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{2\sigma^2} \sum_{i=1}^{n} \left(y^{(i)} - \theta^T x^{(i)}\right)^2$$

since $n \log \frac{1}{\sqrt{2\pi}\sigma}$ is a constant, maximize $L(\theta)$ is equivelant to $\min_{\theta} \frac{1}{2} \sum_{i=1}^{n} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$)