Learning algorithms, linear vs. quadratic

supervised learning = "right answers" given

regression = predict continuous valued output (price)

classification = discrete valued output (0 or 1), could be 1→N (must be discrete)

infinite # of features, other algorithm

unsupervised learning, not told what to do, given dataset find structure

clustering algorithm = find clusters, used to organize large computer clusters

cocktail party problem = find structure and separate sounds (audio processing), 1 line of code

[W,s,v]=svd((repmat(sum(x.*x,1),size(x,1),1).*x)*x');

Octave (open-source) or MatLab

Arthur Samuel (1959): Machine Learning: Field of study that gives computers the ability to learn without being explicitly programmed

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

Machine Learning algorithms:

- Supervised Learning
- Unsupervised Learning

Others: Reinforcement Learning, recommender systems

Model and Cost Function / Linear Regression w/ 1 Variable

regression = predict real-valued output / classification = discrete-valued output

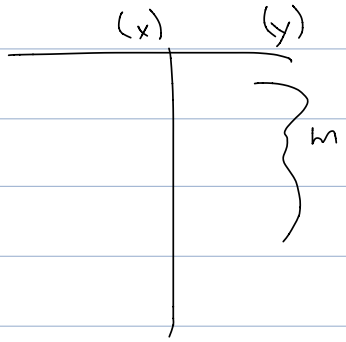
m = Number of training examples

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

↓

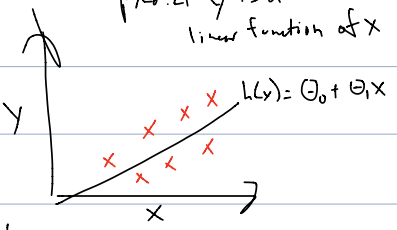
Size of house (x) → h (hypothesis) → Estimated Price (estimated value of y)

How do we represent h?

$h_\theta(x) = \theta_0 + \theta_1 x$

Shorthand: $h(x)$

h maps from x's to y's



Linear regression with one variable (x)

Univariate linear regression

└ one variable

Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x$

θ_i's: Parameters

How to choose?

$(x^{(i)}, y^{(i)})$

Idea: Choose θ_0, θ_1 so that $h_\theta(x)$ is close to y for our training examples (x, y)

$$\underset{\theta_0, \theta_1}{\text{minimize}} \; \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

m = # of training examples

$h_\theta(x^{(i)}) = \theta_0 + \theta_1 x^{(i)}$

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

$$\underset{\theta_0, \theta_1}{\text{minimize}} \; J(\theta_0, \theta_1)$$

$J(\theta_0, \theta_1)$: cost function

Goal

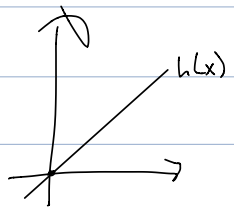
Squared error function

Simplified $h_\theta(x) = \theta_1 x$

$\theta_0 = 0$

$J(\theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$

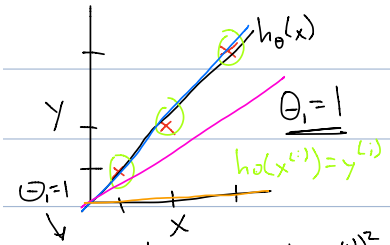
$\theta_1 x^{(i)}$



minimize $J(\theta_1)$
θ_1

$h_\theta(x)$

(for fixed θ_1, this is a function of x)

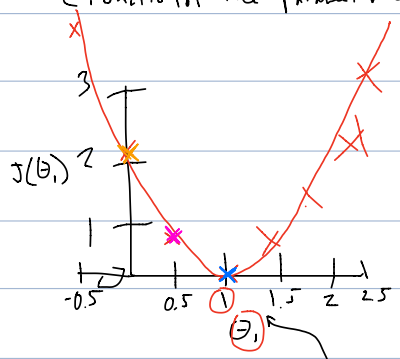


$J(\theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$

$= \frac{1}{2m} \sum_{i=1}^{m} (\theta_1 x^{(i)} - y^{(i)})^2 = \frac{1}{2m}(0^2 + 0^2 + 0^2) = 0^2$

$J(\theta_1)$

(function of the parameter θ_1)



minimize $J(\theta_1)$
θ_1

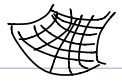
$J(1) = 0$ $J(0.5) = 0.58$

$J(0) = 14/6$

$\frac{1}{2m}[(1)^2 + (2)^2 + (3)^2] = \frac{14}{2 \cdot 3}$

1 parameter θ_1 = bow

2 parameters θ_0 & θ_1 = bow (contour)



Contour plots ≈ Contour figures

Parameter Learning / Gradients

Gradient descent (linear regression, and other stuff)

Have some function $J(\theta_0, \theta_1)$ $J(\theta_0, \theta_1, \theta_2, \ldots, \theta_n)$

Want $\min J(\theta_0, \theta_1)$ $\min J(\theta_0, \ldots, \theta_n)$

θ_0, θ_1 $\theta_0 \cdots \theta_n$

Outline

- Start with some θ_0 θ_1 (say $\theta_0 = 0, \theta_1 = 0$)
- Keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$ until hopefully, end up at a minimum (local)

Gradient descent algorithm

Assignment: $a := b$ $a := a+1$ | Truth Assertion: $a = b$ ← same

repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \quad (\text{for } j = 0 \text{ and } j = 1)$$

}

α: learning rate (step size) — $\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$: derivative term

simultaneously update θ_0 AND θ_1

Correct: Simultaneous update (natural)

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

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

$\theta_0 := temp0$

$\theta_1 := temp1$

Incorrect (Not gradient descent, different algo)

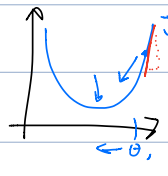
$temp0 := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$

$\theta_0 := temp0$

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

$\theta_1 := temp1$

$\min_{\theta_1} J(\theta_1)$ $\theta_1 \in \mathbb{R}$



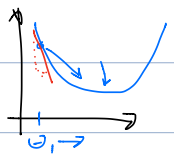
$J(\theta_1)$ $(\theta_1 \in \mathbb{R})$

$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1)$

$\frac{d}{d\theta_1} J(\theta_1) \geq 0$

$\theta_1 := \theta_1 - \alpha$ (positive number)

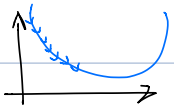
always(+)



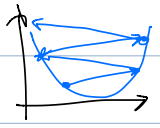
$\frac{d}{d\theta_1} J(\theta_1) \leq 0$

$\theta_1 := \theta_1 - \alpha$ (negative number)

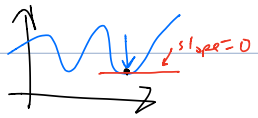
If α is too small, gradient descent can be slow



If α is too large, gradient can overshoot minimum. It may fail to converge, or even diverge.

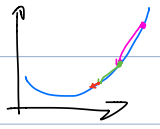


If at local optimum, derivative is 0, so don't move



Gradient descent can converge to local minimum, even with the learning rate α fixed

As we approach a local minimum, gradient descent will automatically take smaller steps. So, no need to decrease α over time



Any cost function J

Apply gradient descent to Linear Regression Model

Need partial derivative term

$$\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_j} \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

$$= \frac{\partial}{\partial \theta_j} \frac{1}{2m} \sum_{i=1}^{m} \left(\theta_0 + \theta_1 x^{(i)} - y^{(i)}\right)^2$$

θ_0 $j=0$: $\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)$

θ_1 $j=1$: $\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) \cdot x^{(i)}$

Repeat until convergence {

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

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

update θ_0 and θ_1 simultaneously

}

Cost function is always a "convex" function (bowl shape)

always get global optimum

Usually initialize $0, 0$

"Batch" Gradient Descent

"Batch": Each step of gradient descent uses all the training examples

$\hookrightarrow \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$