Start simple w/ $\boxed{\underline{\text{Linear Regression}}}$

Let's continue using house price example but use an input or feature in $\mathbb{R}^2$ - so sq ft and number of bedrooms.

so x's $\in \mathbb{R}^2$ and for notation

$x_1^{(i)}$ = sq ft of house (i)

$x_2^{(i)}$ = # of bedrooms in house (i)

Now, want to decide what should look like.

Here, we are doing linear regression (really affine) in x.

i.e. $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x_2$

where θ_i are the parameters in our models (sometimes called weights)

so regression here means choosing the "best" weights

Note $h_\theta(x) : \mathcal{X} \to \mathcal{Y}$
$\mathbb{R}^2 \to \mathbb{R}$

often will write h, not h_θ, sometimes even $h(x;\theta)$.

Depends on if we want to emphasize parameter dependence.

Also, for linear regression sometimes introduce

$x_0 = 1$ (intercept form)

d ← # of input variables, not counting intercept one

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

So how do we choose the θ_i?

will do dumb but generalizable way, then linear algebra way.

The θ_i's depend on how we define a good fit.

Have some sense $h(x)$ should be close to y for our training set — seems reasonable.

Thus, we write down a measure of goodness of fit.

Call it:

- Loss function
- Cost
- Energy
- Error,

Etc.

Arbitrary Choice:

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

And the goal is to find Θ to make this as small as possible.

(This special choice of J called $\boxed{\underline{\text{ordinary least squares}}}$)

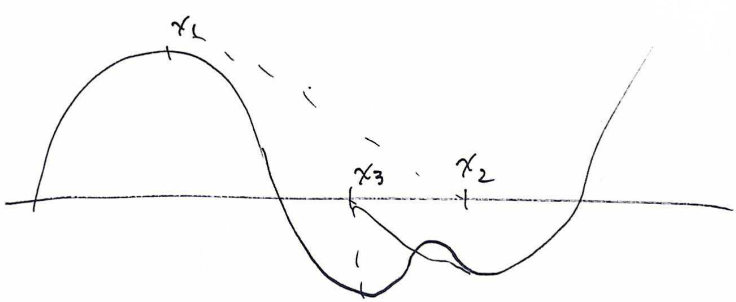
so want Θ to minimize J

or Θ that satisfies $\underset{\Theta}{\operatorname{argmin}} J(\Theta) = \Theta$

Argument That Minimizes.

Want to describe an algorithm to find Θ:

start w/ initial guess for Θ and use algorithm that changes Θ. slowly hopefully making J smaller at each step.



Gradient Descent

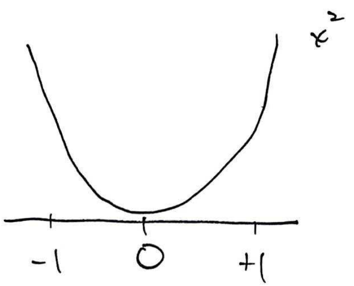
start w/ Θ guess and want formula for next $\tilde{\Theta}$.

$$\tilde{\Theta}_j = \Theta_j - \alpha \frac{\partial}{\partial \Theta_j} J(\Theta)$$

This is update for j^{th} component but would update Θ_j simultaneously

Basically, - - - is a gradient: $\tilde{\Theta} = \tilde{\Theta} - \alpha \nabla J(\Theta)$.

Interestingly, α is called the learning rate.



$$\nabla J(x) = 2x$$

$$-\alpha 2$$

$$1 - 1 \cdot 2$$

The gradient of J gives the direction in which $J(\theta)$ is increasing the fastest, so $-\nabla J(\theta)$ is steepest descent.

Thus, we need to figure out $\nabla J(\theta)$.

$\boxed{\text{Gradient Descent}}$ for Least Squares

$$\tilde{\Theta} = \Theta - \alpha \nabla J(\Theta)$$

Need to figure out $\nabla J(\Theta)$

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

$$h_\Theta(x) = \sum_{i=0}^{d} \Theta_i x_i = \Theta^T x$$

$x_0 = 1$

Start w/ $n = 1$, so no sum in J

training set is (x, y)

$$\frac{\partial}{\partial \Theta_j} J(\Theta) = \frac{\partial}{\partial \Theta_j} \frac{1}{2}\left(h_\Theta(x) - y\right)^2$$

$$= 2 \cdot \frac{1}{2}\left(h_\Theta(x) - y\right) \frac{\partial}{\partial \Theta_j}\left(h_\Theta(x) - y\right)$$

$$= \left(h_\Theta(x) - y\right) \frac{\partial}{\partial \Theta_j}\left(\sum_{i=0}^{d} \Theta_i x_i - y\right)$$

$$= \left(h_\Theta(x) - y\right) x_j$$

If only have one training example,

then the update is

$$\tilde{\Theta}_j = \Theta_j + \alpha \left(y^{(i)} - h_\Theta(x^{(i)}) \right) x_j^{(i)}$$

Note: magnitude of update $\underset{\text{proportional to}}{\propto}$ error term $\left(y^{(i)} - h_\Theta(x^{(i)}) \right)$

If your prediction nearly predicts value of $y^{(i)}$,

then little update.

What about more data? "Σ" is linear or is

the derivative

$$\tilde{\Theta}_j = \Theta_j + \alpha \sum_{i=1}^{n} \left(y^{(i)} - h_\Theta(x^{(i)}) \right) x_j^{(i)}, \quad \forall j$$

OR

$$\underbrace{\tilde{\Theta}}_{\text{vectors}} = \underbrace{\Theta}_{\text{vectors}} + \overbrace{\alpha}^{\text{scalars}} \sum_{i=1}^{n} \overbrace{\left(y^{(i)} - h_\Theta(x^{(i)})\right)}^{\text{scalars}} \underbrace{x^{(i)}}_{\text{vectors}}$$

called batch gradient descent

- uses all training data in every step for update to the parameters $\tilde{\Theta}$.

$$h_\Theta(x) : \underbrace{\mathcal{X}}_{\mathbb{R}^n} \to \overbrace{\mathcal{Y}}^{\mathbb{R}}$$

J is in this case convex so should have a unique solution & gradient descent should converge.

Can modify gradient in various ways to get good optimization schemes.

As an alternative, we can use the single update rule but w/ more data.

for $i = 1, \ldots, n$

$$\tilde{\Theta} = \Theta + \alpha \left(y^{(i)} - h_\Theta \left(x^{(i)} \right) \right) x^{(i)}$$

Repeat until whole thing has converged.

So we just keep updating $\tilde{\Theta}$ one data point at a time - called

$\boxed{\text{Stochastic Gradient Descent}}$

(incremental)

Update as data is introduced instead of using all the data.