LECTURE 2: LINEAR REGRESSION AND GRADIENT DESCENT

Outline:
- Linear Regression
- Batch and Stochastic Gradient Descent
- Normal Equations

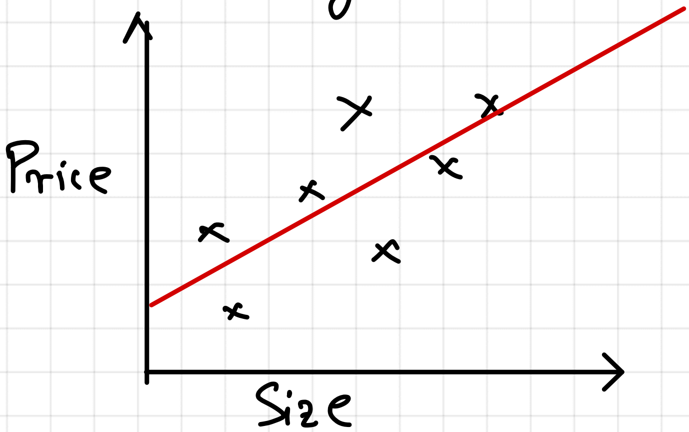
Supervised Learning → Regression
Supervised Learning → Classification

---

* Linear Regression:

Q. Calculating House Price using Size an

Fit a line to this data ←



Supervised Learning:

i/p size of house ↓

Training Set → Learning Algorithm → Hypothesis h "Output" → Estimated Price.

Design choices in ML projects:

1. What is the workflow?
2. What is the dataset?
3. How to represent the hypothesis?

Hypothesis for Linear Regression:

Lets add another feature to our problem:
# of bedrooms
↓
X_2

$$\boxed{h(x) = \theta_0 + \theta_1 x}$$ ← Input feature (Size in our example)

If multiple features, then:

$$\boxed{h(x) = \sum_{j=0}^{n} \theta_j x_j}$$ ... where $X_0 = 1$

Notation:

(Used throughout the notes.)

For future reference.

θ: Parameters of the learning algo.
↓
Gets updated by learning algorithm

m: number of training examples/rows

x: "Inputs"/ features

y: Output / target.

(x, y): 1 training example

(x^i, y^i): i^{th} training example

n: no. of features

→ For our problem:

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix} \qquad X = \begin{bmatrix} X_0 \\ X_1 \\ X_2 \end{bmatrix}$$

$$h(x) = \sum_{j=0}^{2} \theta_j X_j$$

where $X_0 = 1$

Because of X_0 & θ_0, X and θ are $n+1$ dimensional features.

→ So how do you choose Theta?

Choose θ so that $h(x)$ is as close to y as possible.

So we write: $h_\theta(x)$ because it depends on both θ and x, although either is
$\|$ fine.
$h(x)$

→ Ordinary Least Squares (OLS) Regression.

For correct/accurate/close predictions on the training set:

$$J(\theta) \Rightarrow \boxed{\underset{\theta}{\text{minimize}} \; \frac{1}{2} \sum_{i=1}^{m} (h_\theta(x) - y)^2}$$

$\frac{1}{2}$: Simplicity for math, because we take derivative later

$h_\theta(x)$: Algorithm prediction

y: True value/price of the house

This equation is the cost function $J(\theta)$, we find parameters "θ" that minimize the cost function $J(\theta)$.

Gradient Descent

↳ Minimizes $J(\theta)$

① Start with some value of θ. (Say $\theta = \vec{0}$)

② Keep changing θ to reduce $J(\theta)$.

$$\boxed{\theta_j := \theta_j - \underbrace{\alpha}_{\text{learning rate (hyper-parameter)}} \frac{\partial}{\partial \theta_j} J(\theta)}$$

← One step of Gradient Descent

θ_j → $j = 0, 1, 2 \ldots n$

Note: we use $-\alpha$ because if we used "$+\alpha$" it would move away from convergence.

The direction of slope automatically compensates for the "-ve"

$$\boxed{d(f(x)) \longrightarrow \text{defines direction of steepest descent}}$$

Partial derivative of cost function

We will assume for 1 training example, (however it is summed over all training examples.)

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

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

$$= (h_\theta(x) - y) \frac{\partial}{\partial \theta_j} (\theta_0 + \theta_1 x_1 + \ldots - y)$$

All terms will be zero, except term corresponding to $j \Rightarrow \theta_j x_j$

e.g if $j=1$ then $\theta_1 x_1$

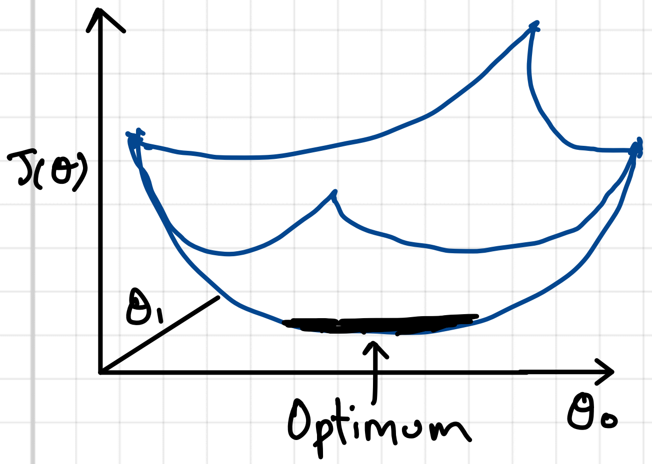
$$\boxed{\therefore \frac{\partial}{\partial\theta_j} J(\theta) = (h_\theta(x) - y)x_j}$$

One step of Gradient Descent is:

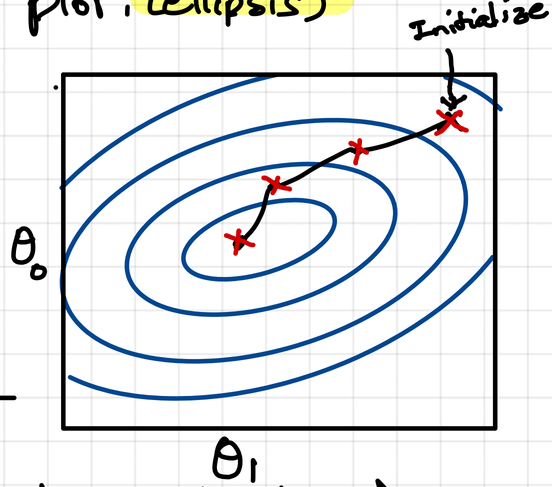
$$\boxed{\theta_j := \theta_j - \alpha \sum_{i=1}^{m} (h_\theta(x^i) - y^{(i)})\, x_j^{(i)}}$$

For each iteration of gradient descent, carry out the update till $\underbrace{j=1,2\dots n}_{\Downarrow\ \text{no. of features}}$,

→ Repeat Gradient Descent until convergence.
$J(\theta)$ for linear regression is a quadratic function. So no local optima, only global optima. (Imagine a big bowl)



Alternatively look at the contours of the plot. (ellipsis)



Step size: depends on α.

$\alpha \Rightarrow$ too large $\rightarrow$ overshoots minima.

$\alpha \Rightarrow$ too small $\rightarrow$ Takes too long to converge

Try a few values.

In contours, direction of steepest descent is always at 90°/orthogonal to contour direction

Gradient Descent

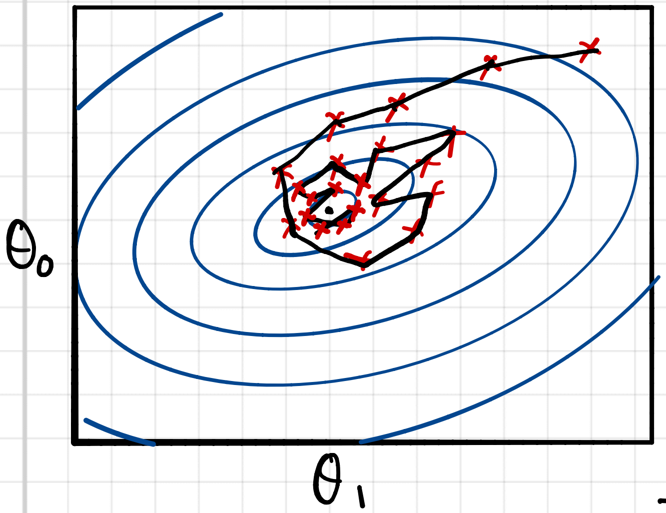
Batch Gradient Descent

- Looks at training data as one batch
- Disadvantage is that each step is very slow because we scan over entire data for each training step
- Very expensive.

Stochastic Gradient Descent

- Update parameters on every training example, instead of the full batch.
- Noisy and much more random, because your model is always oscillating between each training example.
- Never really converges
- However, much more practical in large datasets (thousands, millions)
- You end up pretty close to the global optimum, and get good enough predictions.

Stochastic Gradient Descent



★ In practice with SGD, we slowly decrease learning rate over time, so you take smaller and smaller steps and end up really, really close to global optimum.

- For linear regression (only): another method to get straight to global optimum without iterative algorithm.

Normal Equations.

$J(\theta) \rightarrow$ function mapping from parameters to real no.s.

Matrix Derivative

$\nabla_\theta J(\theta)$ $\quad \theta \in \mathbb{R}^{n+1} \rightarrow$ (n+1 dimensional vector)

$\hookrightarrow$ Derivitive of J w.r.t θ

In our problem: $\nabla_\theta J(\theta) = \begin{bmatrix} \frac{\partial J}{\partial \theta_0} \\ \frac{\partial J}{\partial \theta_1} \\ \frac{\partial J}{\partial \theta_2} \end{bmatrix}$

$\Downarrow$

$\theta \in \mathbb{R}^{2+1}$

Matrix Derivative

e.g. $\nabla_A f(A)$ $\quad A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$

$f(A) = A_{11} + A_{12}^2$

$$\nabla_A f(A) = \begin{bmatrix} \frac{\partial f(A)}{\partial A_{11}} & \frac{\partial f(A)}{\partial A_{12}} \\ \frac{\partial f(A)}{\partial A_{21}} & \frac{\partial f(A)}{\partial A_{22}} \end{bmatrix} = \begin{bmatrix} 1 & 2A_{12} \\ 0 & 0 \end{bmatrix}$$

Steps: $\nabla_\theta J(\theta) \overset{\text{set}}{=} 0 \rightarrow$ Solve for θ

Trace of A ($\mathrm{tr}A$) = Sum of diagonal elements $= \sum A_{ii}$

Properties of trace:

$\boxed{\begin{array}{l|l} \text{(1) } \mathrm{tr}(A) = \mathrm{tr}(A^T) & \text{(3) } \mathrm{tr}AB = \mathrm{tr}BA \\ \text{(2) } f(A) = \mathrm{tr}AB & \mathrm{tr}ABC = \mathrm{tr}CAB \text{ (cyclic rotation)} \\ \nabla_A f(A) = B^T & \text{(4) } \nabla_A \mathrm{tr}AA^TC = CA + C^TA \end{array}}$

Solve:

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

$$X = \begin{bmatrix} — (x^1)^T — \\ — (x^2)^T — \\ \vdots \\ — (x^m)^T — \end{bmatrix} \quad \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix}$$

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

$$\vec{y} = \begin{bmatrix} y^1 \\ y^2 \\ \vdots \\ \vdots \\ y^m \end{bmatrix}$$

So now,

$$J(\theta) = \frac{1}{2}\underbrace{\underbrace{(X\theta - y)}^T (X\theta - y)}_{zz^T = \Sigma z^2}$$

$$\begin{bmatrix} h_\theta(x^1) - y^1 \\ \vdots \\ \vdots \\ h_\theta(x^m) - y^m \end{bmatrix}$$

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

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

$$= \frac{1}{2}\nabla_\theta\left[\underline{\theta^T X^T X\theta} - \underline{\theta^T X^T y} - \underline{y^T X\theta} - \underline{y^T y}\right]$$

Take derivative of each term

$$= \frac{1}{2}\left[X^T X\theta + X^T X\theta - X^T y - X^T y\right]$$

Refer lecture notes for derivatives

$$= X^T X\theta - X^T y = \vec{0}$$

Normal equation

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

$$\boxed{\therefore \theta = (X^T X^{-1})(X^T Y)}$$

One step solution for parameters

Normal Equations.