* NumPy, Matplotlib, Pandas & other necessary libraries.

NumPy import numpy as np

ones = np.ones(shape), dtype=int); → Creates an array of one with desired shape

** np.hstack((array_we_want_to_stack, target_array));
↳ hstack takes a tuple of arrays as arguments)

alternatives:

① np.c_[ones, original_array]; → Does column wise concatenation

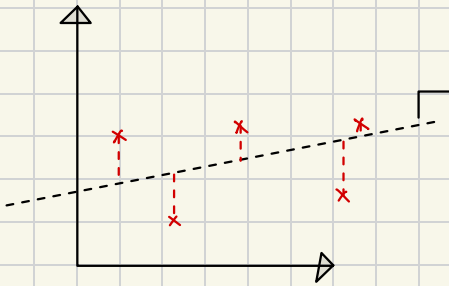
② np.insert(original_array, index, value, axis=1); → Most control.

Pandas import pandas as pd;

df = pd.read_csv("file_name", header=None, sep=",");
loads without headers

columns = ['a', 'b', 'c', 'd', 'e', 'f']
df.columns = columns → Inserts columns

Linear Regression with GD



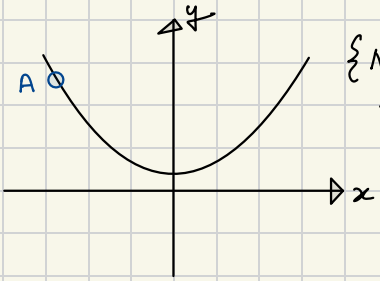
error = y − guess

(input, output)

x y

Cost: $\sum_{i=0}^{n} (\hat{y}_i - y_i)^2$ → We want to minimise this function.

$\therefore$ Cost = $\sum (\hat{y}_i - y_i)^2$ this is kind of like $y = x^2$



{ Now, we need to find the value of x for that y is lowest.

Suppose, we're at point A & if we want to reduce the function, we need to find its gradient. But how do we go down?

→ We calculate the slope.

$m = m + \Delta m$

$b = b + \Delta b$

$\therefore J(m,b) = \frac{1}{n} \sum_{i=0}^{n} (mx + b - y)^2$

$\frac{\delta}{\delta m} J = 2(mx + b - y)m$ & $\frac{\delta}{\delta b} J = 2(mx + b - y)(1)$

* $\frac{1}{n}$ in $J(m,b) = \frac{1}{n}\sum(\hat{y}_i - y_i)$?

→ We want to take the average error & minimise it.

→ Makes it scale irrelevant so that same learning rate works in different datasets.

We also sometimes take $\frac{2}{n}$, this is purely because of algebraic convenience.

Update rule: $m^{(t+1)} = m - \alpha \frac{\partial}{\partial m} J$

$b^{(t+1)} = b - \alpha \frac{\partial}{\partial b} J$

* Example:

$(x, y) = \{(1,2), (2,3), (3,5)\}$ predict y when $x = 7$

here, $n = 3$, model, $\hat{y} = mx + b$ let, $\hat{y}_1 = \hat{y}_2 = \hat{y}_3 = 0$ { because, we assumed $m = 0$ & $b = 0$ }

$$J = \frac{1}{n}\sum_0^n (\hat{y}_i - y_i)^2 = \{(0-2)^2 + (0-3)^2 + (0-5)^2\}/3$$

$$= \frac{4+9+25}{3} = \frac{38}{3} \approx 12.67$$

let, $\alpha = 0.01$

Now, we'll minimise loss by updating m & b

Given, $J = \frac{1}{n}\sum_0^n (mx + b - y_i)^2$

$$\therefore \frac{\partial J}{\partial m} = \frac{1}{n}\sum(mx_i + b - y_i) \times 2 \times (x_i)$$

$$\& \quad \frac{\partial J}{\partial x} = \frac{2}{n}\sum(mx_i + b - y_i)(1)$$

$m^{(t+1)} = m^t + \alpha \frac{\partial}{\partial m} J$

$b^{(t+1)} = b^t + \alpha \frac{\partial}{\partial b} J$

1st step,

$$\frac{\partial J}{\partial m} = \frac{1}{3}\{(0 \times 1 + 0 - 2) \times 2(1) + (0 \times 2 + 0 - 3)(2 \times 2) + (-5)(2 \times 3)\} = -15.33$$

$$\therefore m^1 = m^0 - \alpha\left(\frac{\partial J}{\partial m}\right) = 0 - 0.01(-15.33) = 0.1533$$

Let's get b^1

$$\frac{\partial J}{\partial b} = \frac{2}{n}\sum_{0}^{n}(mx_i + b - y_i) = \frac{2}{3}\left(-2-3-5\right) = -6.667$$

$$\therefore b^1 = b^0 - \alpha\left(\frac{\partial J}{\partial b}\right) = 0 - 0.0667 = 0.0667$$

So, model updates to,

$$\hat{y}_i = 0.1533\,x_i + 0.0667$$

$$\therefore \hat{y}_1 = 0.22 \quad \hat{y}_2 = 0.3733 \quad \hat{y}_3 = 0.5266$$

Current error: $J = \frac{1}{n}\sum_{0}^{n}(\hat{y}_i - y_i)^2$

$$= \frac{1}{3}\left((0.22-2)^2 + (0.3733-5)^2 + (0.5266-5)^2\right)$$

$$= \left\{(-1.78)^2 + (-4.6267)^2 + (-4.4734)^2\right\}/3$$

$$= 10.022067$$

So, we've reduced error in step 1 but we have two more steps to go. It took a lot of manual calculations, let's simplify a bit using matrices.

* Dataset $\{(1,2), (2,3), (3,5)\}$ & prediction $\hat{y}_i = mx_i + b$

$\theta = \begin{pmatrix} m \\ \theta \end{pmatrix}$ so, $\overline{\hat{y}} = \overline{x}\,\overline{\theta}$

* Design matrix:

$$\overline{x} = \begin{pmatrix} 1 & 1 \\ 2 & 1 \\ 3 & 1 \end{pmatrix} \quad \overline{\theta} = \begin{pmatrix} m \\ \theta \end{pmatrix} \quad \overline{\hat{y}} = \overline{x}\,\overline{\theta} = \begin{pmatrix} 1 & 1 \\ 2 & 1 \\ 3 & 1 \end{pmatrix}\begin{pmatrix} m \\ \theta \end{pmatrix}$$

$$= \begin{pmatrix} m+\theta \\ 2m+\theta \\ 3m+\theta \end{pmatrix}$$

as $m, \theta = 0, 0$ $\quad n = 3$

$\therefore \bar{\hat{y}} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$ $\quad$ Given, $\bar{y} = \begin{pmatrix} 2 \\ 3 \\ 5 \end{pmatrix}$

$\bar{e} = \bar{\hat{y}} - \bar{y} = \begin{pmatrix} -2 \\ -3 \\ -5 \end{pmatrix}$ $\quad \bar{e}^2 = (\bar{\hat{y}} - \bar{y})^2 = \|\bar{\hat{y}} - \bar{y}\|_2$

$= (\bar{\hat{y}} - \bar{y})^T (\bar{\hat{y}} - \bar{y})$

$= (2 \;\; 3 \;\; 5) \begin{pmatrix} 2 \\ 3 \\ 5 \end{pmatrix}$

$= (4 + 9 + 25) = 38$

* Cost Function,

$J = \frac{\bar{e}^2}{n} = 38/3 = 12.667$

* Gradient,

$J(\theta) = 1/n \left(\bar{\hat{y}} - \bar{y}\right)^T \left(\bar{\hat{y}} - \bar{y}\right) = \frac{1}{n} (\bar{x}\theta - \bar{y})^T (\bar{x}\theta - \bar{y})$

$= \frac{1}{n} (\bar{x}^T\theta^T - \bar{y}^T)(\bar{x}\theta - \bar{y})$

$= \frac{1}{n} (\theta^T\bar{x}^T\bar{x}\theta - \bar{x}^T\theta^T\bar{y} - \bar{y}^T\bar{x}\theta - \bar{y}^T\bar{y})$

$= \frac{1}{n} (\theta^T\bar{x}^T\bar{x}\theta - 2\bar{x}^T\theta^T\bar{y} - \bar{y}^T\bar{y})$

$J(\theta) = \frac{1}{n} (\theta^T\bar{x}^T\bar{x}\theta - 2\bar{y}^T\bar{x}\theta - \bar{y}^T\bar{y})$

* Matrix Calculus Identities

1. $\nabla_\theta (\theta^T A \theta) = 2A\theta$ {When A is symmetric & if $A = x^Tx$ then A is always symmetric}
2. $\nabla_\theta (c^T\theta) = c$
3. $\nabla_\theta (\text{constant}) = 0$

→ Appendix 1

* Applying these in the cost function,

1. $\nabla_\theta (\theta^T\bar{x}^T\bar{x}\theta) = 2\bar{x}^T\bar{x}\theta$
2. $\nabla_\theta (2\bar{y}^T\bar{x}\theta) = 2\bar{x}^Ty$
3. $\nabla_\theta (\bar{y}^T\bar{y}) = 0$

$\therefore \nabla_\theta J = (2\bar{x}^T\bar{x}\theta - 2\bar{x}^T\bar{y})/n$

$= \frac{2\bar{x}^T}{n} \underbrace{(\bar{x}\theta - \bar{y})}_{\text{Error Vector}}$

↑ Gradient (w.r.t m & b)

Update rule : $\theta := \theta - \alpha \nabla_\theta J$

Let's verify !

$\{(1,2),(2,3),(3,5)\}$

$$\underset{A}{\bar{x}} = \begin{pmatrix} 1 & 1 \\ 2 & 1 \\ 3 & 1 \end{pmatrix} \quad \underset{B}{\bar{\theta}} = \begin{pmatrix} m \\ b \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \quad \underset{C}{\bar{y}} = \begin{pmatrix} 2 \\ 3 \\ 5 \end{pmatrix}$$

$$\hat{y} = \bar{x}\bar{\theta} \qquad y_1 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

$$\bar{e} = \hat{y} - y$$

$$\bar{e} = \begin{pmatrix} -2 \\ -3 \\ -5 \end{pmatrix}$$

*Cost Calc

$$J(\theta) = \frac{1}{n}(\bar{x}\theta - y)^T(\bar{x}\theta - y)$$

$$= \frac{1}{3}\begin{pmatrix} -2 & -3 & -5 \end{pmatrix}\begin{pmatrix} -2 \\ -3 \\ -5 \end{pmatrix} = \frac{1}{3}\left(2^2 + 3^2 + 5^2\right) = 38/3 = 12.67$$

$$* \nabla_\theta J(\theta) = \frac{2}{n}\bar{x}^T(\bar{x}\theta - y)$$

$$= \frac{2}{3}\begin{pmatrix} 1 & 2 & 3 \\ 1 & 1 & 1 \end{pmatrix}\left(\begin{pmatrix} 1 & 1 \\ 2 & 1 \\ 3 & 1 \end{pmatrix}\begin{pmatrix} 0 \\ 0 \end{pmatrix} - \begin{pmatrix} 2 \\ 3 \\ 5 \end{pmatrix}\right)$$

$$= \frac{2}{3}\begin{pmatrix} -23 \\ -10 \end{pmatrix} = \begin{pmatrix} -46/3 \\ -20/3 \end{pmatrix} = \begin{pmatrix} -15.33 \\ -6.667 \end{pmatrix}$$

$$\therefore \nabla_\theta J(\theta) = \begin{pmatrix} -15.33 \\ -6.667 \end{pmatrix}$$

$$\therefore \theta^1 = \theta - \alpha\begin{pmatrix} -15.33 \\ -6.667 \end{pmatrix} \quad \{\alpha = 0.01\}$$

$$= \begin{pmatrix} 0.1533 \\ 0.0667 \end{pmatrix}$$

$$\hat{y}_1 = \bar{x}\,\bar{\theta} = \begin{pmatrix} 1 & 1 \\ 2 & 1 \\ 3 & 1 \end{pmatrix}\begin{pmatrix} 0.1533 \\ 0.0667 \end{pmatrix} = \begin{pmatrix} 0.22 \\ 0.3733 \\ 0.5266 \end{pmatrix}$$

$$e^1 = \begin{pmatrix} -1.78 \\ -2.626 \\ -4.9739 \end{pmatrix}$$

$$\theta^{(2)} = \theta^{(1)} - \alpha\left(\nabla_\theta J(\theta)\right) = \theta^{(1)} - \alpha\left(\frac{2}{n} x^T (x\theta^{(1)} - y)\right)$$

$$= \theta^{(1)} - 0.01\left(\frac{2}{3}\begin{pmatrix} 1 & 2 & 3 \\ 1 & 1 & 1 \end{pmatrix}\left(\begin{pmatrix} 1 & 1 \\ 2 & 1 \\ 3 & 1 \end{pmatrix}\begin{pmatrix} 0.1533 \\ 0.0667 \end{pmatrix} - \begin{pmatrix} 2 \\ 3 \\ 5 \end{pmatrix}\right)\right)$$

$$= \theta^{(1)} - 0.01\left(\frac{2}{3}\begin{pmatrix} 1 & 2 & 3 \\ 1 & 1 & 1 \end{pmatrix}\begin{pmatrix} -1.78 \\ -2.626 \\ -4.9739 \end{pmatrix}\right) e^1$$

$$\therefore \theta^{(2)} = \begin{pmatrix} 0.289698 \\ 0.125896 \end{pmatrix}$$

$e = x\theta - y$

$$\theta^3 = \theta^{(2)} - 0.01 \times \frac{2}{3}\begin{pmatrix} 1 & 2 & 3 \\ 1 & 1 & 1 \end{pmatrix}\begin{pmatrix} -1.584 \\ -2.274 \\ -4.005 \end{pmatrix}$$

$$= \begin{pmatrix} 0.289698 \\ 0.125896 \end{pmatrix} - \begin{pmatrix} -0.121263 \\ -0.052563 \end{pmatrix} = \begin{pmatrix} 0.410912 \\ 0.1734 \end{pmatrix}$$

$$J(\theta)^{(3)} = \left|\left(\bar{x}\,\theta^{(3)} - y\right)\right|_2 \times 1/n \quad (n=3)$$

$$= 6.29$$

* How long will we go?

→ Our error term is reducing, so we can go as low as we want till error ≈ 0

But, we can assign few other stopping conditions

1. if $J(\theta) \approx 0$
2. iteration count limit
3. $\|e\|_2 \approx 0$
4. $\| J(\theta)^{t+1} - J(\theta)^{(1)} \| < \epsilon$ or ≈ 0

But no matter what we do, we need computational power because datasets can become very large!

→ We'll use python to create our linear regress with GD algorithm, calculate accuracy.

* Appendix 1

① $\theta = \begin{pmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_d \end{pmatrix} \in \mathbb{R}^{d \times 1}$ $\quad \nabla_\theta f(\theta) = \begin{pmatrix} \partial f/\partial \theta_1 \\ \partial f/\partial \theta_2 \\ \vdots \\ \partial f/\partial \theta_d \end{pmatrix}$

* $\frac{d}{d(\text{vector})}(\text{vector}) = (\text{vector})$

② $\nabla_\theta(c^T\theta) = c$

$c^T\theta = \sum_{0}^{d} c_i d_i \quad \frac{\partial}{\partial \theta_j}(c^T\theta) = c_j$

Stacking all in C

③ $\nabla_\theta(k) = 0$

Constant rule

④ $\nabla_\theta \theta^T A \theta = 2A\theta$ if A is symmetric.

Linear Regression = Quadratic Optimisation

Convex = Single Global Minima

Normal Equation (for any number of features)

$\theta = (x^T x)^{-1} x^T y$ $\quad m = \frac{\sum(x_i - \bar{x})(y_i - y)}{\sum(x_i - \bar{x})^2}$

Stochastic Gradient Descent

→ Regular GD uses all datapoints to calculate the gradient but this becomes costly over time as n increases.

→ SGD uses one datapoint at a time.

*Steps:

(1) Define Per-Sample Loss:

$$l_i(\theta) = (x_i^T\theta - y_i)^2$$

$$J\theta = \frac{1}{n}\sum_{1}^{n} l_i(\theta)$$

(2) Gradient of One Sample (x_i, y_i)

$$\nabla_\theta l_i(\theta) = 2(x_i^T\theta - y_i)(x_i)$$

(3) Update rule

$$\theta \leftarrow \theta - \alpha \cdot \underbrace{2x_i(x_i^T\theta - y_i)}_{\nabla_\theta l_i(\theta)}$$

Example:

$\hat{y} = mx + b$ $\quad x_i = [x_i, 1]$ $\quad \theta = \begin{bmatrix} m \\ b \end{bmatrix}$

i	x	y
1	1	2
2	2	3
3	3	5

$\theta^{(0)} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$

$\alpha = 0.1$

One random point $(x_2, y_2) = (2, 3)$

Design Matrix $x_2 = \begin{pmatrix} 2 & 1 \end{pmatrix}$

$$\hat{y}_2 = x_2\theta = \begin{pmatrix} 2 & 1 \end{pmatrix}\begin{pmatrix} 0 \\ 0 \end{pmatrix} = 0 + 0 = 0$$

$$e_2 = 0 - 3 = -3 = x_2\theta - y_2$$

G = Gradient $= \nabla_\theta l_2(\theta) = 2x_2^T e_2$

$$= 2\begin{pmatrix} 2 \\ 1 \end{pmatrix}(-3) = \begin{pmatrix} -12 \\ -6 \end{pmatrix}$$

Update: $\theta^{(1)} = \theta - \alpha G = \begin{pmatrix} 0 \\ 0 \end{pmatrix} - (0.1)\begin{pmatrix} -12 \\ -6 \end{pmatrix} = \begin{pmatrix} 1.2 \\ 0.6 \end{pmatrix}$

*2nd Iteration {Random point $(x_1, y_1) = (1, 1)$ Design Matrix $\begin{pmatrix} 1 & 1 \end{pmatrix}$}

$x_1 = (1 \ 1) \quad y_1 = 2$

$$y_1 = (1 \ 1)\begin{pmatrix} 1.2 \\ 0.6 \end{pmatrix} = 1.2 + 0.6 = 1.8$$

$$e = 1.8 - 2 = -0.2$$

$$G = 2x_1^T e = 2\begin{pmatrix} 1 \\ 1 \end{pmatrix}(-0.2) = \begin{pmatrix} -0.4 \\ -0.4 \end{pmatrix}$$

$$\theta^{(2)} = \theta^{(1)} - \alpha G$$

$$= \begin{pmatrix} 1.2 \\ 0.6 \end{pmatrix} - 0.1\begin{pmatrix} -0.4 \\ -0.4 \end{pmatrix} = \begin{pmatrix} 1.24 \\ 0.69 \end{pmatrix}$$

*Although S.GD never truly settles

$\alpha_t = \frac{\alpha_0}{1 + kt}$ so we need to sometimes implemt learning rate decay.

Takeaways

1. SGD uses one sample.
2. Each update is cheap but noisy.
3. Over many updates noise averages out.
4. Learning rate must be smaller than VGD.

Mini Batch Gradient Descent

→ Similar concept as VGD but this time we work with a sub-sample.

Algorithm: ① Select sample size.
② Select sub-dataset randomly.
③ Find gradient of that sub-dataset.
④ Shuffle dataset.
⑤ Found desired expected loss? (J0). Yes! Stop.
No! → Goto Step 2.

Adaptive Gradient Descent AdaGraD

→ Look we're basically calculating how fast objects move in which direction. We are talking about shape of the loss function. Hessian simply describe that shape.

→ Gradient tells about the direction of the steepest descent, but tells nothing about 1) How curved the surface is.
2) Whether the direction is safe to move.

Two very different surfaces can have the same gradient at a point.

Appendix 2

Suppose a 1D function $f(x) = ax^2$ $\quad a = 100 \rightarrow$ very steep. $\quad a = 0.01 \rightarrow$ very flat.

$f'(x) = 2ax \rightarrow$ This is the gradient

But what determines how fast the gradient changes?

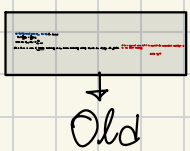
$f''(x) = 2a \rightarrow$ This!

Hessians are basically second derivatives arranged into a matrix.

In gradient descent

$$\theta^{t+1} = \theta^t - \alpha \nabla J(\theta_t)$$

AdaGrad basically gives each parameter its own learning rate based on historical gradients.



Old

① $g_t = \nabla_\theta J(\theta_t) \in \mathbb{R}^d \rightarrow$ Gradient of loss function

② Accumulating squared gradients

$$G_t = \sum_{i=1}^{t} g_i \odot g_i = \overline{g}_t^T \cdot \overline{g}_t$$

$\odot \rightarrow$ Piecewise multiplication

$G_t \in \mathbb{R}^d$

③ Update rule:

$$\theta_{t+1} = \theta_t - \eta \frac{g_t}{\sqrt{G_t} + \epsilon}$$

; Division is elemen-wise.

; ϵ is a small constant (so we don't get $x/0$ error)

Why squared gradients?

i) Makes all contributions positive.

ii) Penalises large gradients disproportionately.

Because

Large gradients → Dominate the sum → shrink step size fast vice-versa.

$H \approx \text{diag}\left(\sum g^2\right)$ → AdaGrad estimates hessians basically because calculating hessians are 1) Expensive 2) Hard 3) Unstable

Weakness

if $G_t \uparrow$ monotonically then $\frac{1}{\sqrt{G_t}} \downarrow$ & eventually learning rate becomes 0.

Example: (1,2), (2,3), (3,5). Simulate LR with AdaGrad.

→ model: $y = \theta_1 x + \theta_0$ $\quad \theta = \begin{pmatrix} \theta_0 \\ \theta_1 \end{pmatrix}$ $\quad$ Design matrix: $\begin{pmatrix} 1 & 1 \\ 2 & 1 \\ 3 & 1 \end{pmatrix}$ x

$$J(\theta) = \frac{1}{2m}\sum_{i=0}^{m}(\hat{y}^{(i)} - y^{(i)})^2$$

$$= \frac{1}{2m}(\hat{y}-y)^T(\hat{y}-y)$$

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

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

$$= \frac{1}{2m}(\theta^T x^T x\theta - 2y^T x\theta + y^T y)$$

$\theta = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ $\quad \hat{y} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} = x\theta$ $\quad e = \hat{y} - y$

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

$$G_t = g_t^T \cdot g_t$$

$$\theta_{t+1} = \theta_t - \eta\frac{g_t}{\sqrt{G_t} + \epsilon}$$

RMSProp

Why AdaGrad fails ?

① $G_t = \sum_{k=1}^{t} g_u \odot g_u$ & $\theta_t - \eta \frac{g_t}{\sqrt{G_t + \epsilon}}$

G_t increases monotonically

We don't need information about all past gradients rather we need information only about recent curvature.

s_t → Running estimate of curvature magnitude per parameter

$s_{t,j} = \mathbb{E}[g_{t,j}^2]$ "How large are the gradients for parameter θ_j, on average, recently ?

$RMS(g_j) = \sqrt{s_{t,j}}$

$g_t = \nabla_\theta J(\theta_t) = \frac{1}{m} x^T (x\theta_t - y)$

$s_t = \rho s_{t-1} + (1-\rho) g_t \odot g_t$ $\quad \rho \in (0, 1)$

$\theta_{t+1} = \theta_t - \eta \frac{g_t}{\sqrt{s_t + \epsilon}}$ $\quad s_t \in \mathbb{R}$

$J(\theta)$

→ $\theta_{t+1} = \theta_t - \eta \left(\text{diag}\left(\sqrt{s_t}\right)^{-1}\right) g_t$

* RMSProp performs adaptive diagonal second order scaling.

* Numerical Example

$(1,2), (2,3), (3,5)$

$$x = \begin{pmatrix} 1 & 1 \\ 2 & 1 \\ 3 & 1 \end{pmatrix} \quad y = \begin{pmatrix} 2 \\ 3 \\ 5 \end{pmatrix}$$

* initialisation

$\eta = 0.1$ $\quad s_0 = 0$

$\rho = 0.9$

$\epsilon \approx 0$

$$J(\theta) = \frac{1}{n}(x\theta - y)^T(x\theta - y)$$

$$= 1/n\left(\theta^T x^T x\theta - x^T\theta^T y - x\theta y^T - y^T y\right)$$

$$g_t = \nabla_\theta J(\theta) = \left(2x^T x\theta - 2x^T y - 0\right)/n$$

$$= \frac{2}{n} x^T \underbrace{(x\theta - y)}_{\text{Error}}$$

$n = 3$

$$s_t = \rho \cdot s_{t-1} + (1-\rho)\, g_t \odot g_t$$

$$\theta_t = \theta_{t-1} - \eta \frac{g_t}{\sqrt{s_t + \epsilon}}$$

$$g_t = \begin{pmatrix} -15.33 \\ -6.667 \end{pmatrix}$$

$$s_t = \rho \cdot s_{t-1} + (1-\rho)\, g_t \odot g_t$$

$$= 0.9 \times \begin{pmatrix} 0 \\ 0 \end{pmatrix} + (0.1) \begin{pmatrix} (-15.33)^2 \\ (-6.667)^2 \end{pmatrix}$$

$$= \begin{pmatrix} 235.111 \\ 44.449 \end{pmatrix}$$

$$\sqrt{s_t} = \begin{pmatrix} 4.847 \\ 2.108 \end{pmatrix} \qquad \frac{g_1}{s_1} = \begin{pmatrix} -15.33/4.847 \\ -6.667/2.108 \end{pmatrix} = \begin{pmatrix} -3.164 \\ -3.163 \end{pmatrix}$$

$$\theta = \begin{pmatrix} 0 \\ 0 \end{pmatrix} - 0.01 \begin{pmatrix} -3.164 \\ -3.163 \end{pmatrix}$$

* This part follows lecture on Stanford CS229 *
* No need to go through if you've understood the concepts well *

Supervised Learning Setup

- **Regression** (Where does this data fit?) → Discrete
- **Classification** (What type of data is this?) → Classification

$x \rightarrow y$
(Input) (Output)

$n \rightarrow$ number of examples in training set. (x, y)

$d \rightarrow x \in \mathbb{R}^d$ dimension of input.

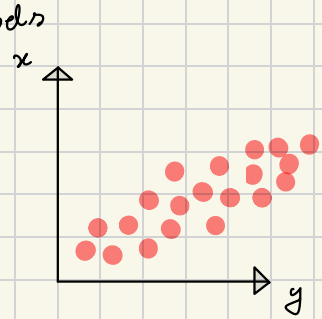
$x^{(i)} \rightarrow$ ith example input.

$y^{(i)} \rightarrow$ ith example output. (label / ground truth)

$(x^{(i)}, y^{(i)}) \rightarrow$ ith example.

* Regression

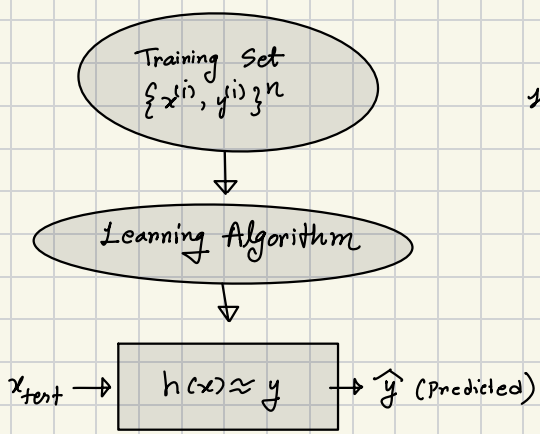
x	y
2104	400
1600	330
2400	369
3000	540



Goal : $f(x) \approx y$ → Learnt Model

Or find the best fit line

* Pipeline



* Linear Regression:

$x \in \mathbb{R}^d$; $y \in \mathbb{R}$, n such examples

Hypothesis:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_d x_d$$

$$h_\theta(x) = \sum_{i=1}^{d} \theta_i x_i + \theta_0 \; ;$$

$x_0 = 1 \rightarrow$ Intercept term

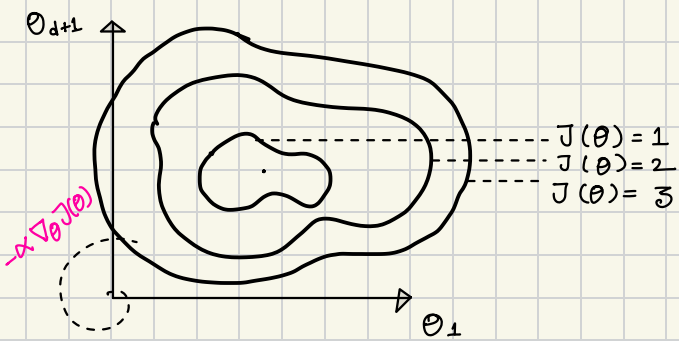
$$h_\theta(x) = \theta_0 x_0 + \dots + \theta_d x_d$$

$$= \sum_{i=0}^{d} x_i \theta_i = \theta^T x$$

Cost Function: $J(\theta) = \frac{1}{2}\sum_{i=1}^{n}(h_\theta(x^{(i)}) - y^{(i)})^2$ | $J(\theta) = f(x)$

$\hat{\theta} \rightarrow \arg\min_{\theta} J(\theta) \rightarrow \arg\min_{\theta} \sum_{i=1}^{n}(h_\theta(x^{(i)}) - y^{(i)})^2$

*Gradient Descent



Contour plot of the cost function

$\theta^{(0)}$ = gnitialisation

$$\theta_j^{(1)} = \theta_j^{(0)} - \alpha\frac{\partial}{\partial\theta_j}J(\theta)$$

$$\therefore \theta^{(1)} = \theta^{(0)} - \alpha\nabla_\theta J(\theta^{(0)})$$

↳ gn vector form

(we repeat this till convergence)

so, $\theta^{(t+1)} = \theta^{(t)} - \alpha\nabla_\theta J(\theta^{(0)})$ → Gradient

Gradient Descent on Linear Regression

$\theta^{(t+1)} = \theta^t - \alpha\nabla_\theta$ [Cost function of 1R]

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

$= \theta^t - \alpha\nabla_\theta\left(\frac{1}{2}\sum_{i=1}^{n}(\theta^T x^{(i)} - y^{(i)})^2\right)$

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

$\theta^{t+1} = \theta^t - \alpha\left(\sum_{i=1}^{n}(\theta^T x^{(i)} - y^{(i)})x^{(i)}\right)$

Scalar Vector Vector

Vector

How to check for convergence

1) $\|\theta^{(t)} - \theta^{(t+1)}\| \approx 0$

or, 2) $\|\nabla_\theta J(\theta^t)\| \approx 0$

or, 3) $|J(\theta^{t+1}) - J(\theta^t)| \approx 0$

$\theta \rightarrow$ Set of all parameters.

*Stochastic Gradient Descent - SGD (I find it stupid)

→ For each small progress we need to scan through all of the dataset.

→ To make it a bit simple we can do,

$$\theta^{(t+1)} = \theta^{(t)} - \alpha \nabla_\theta \tilde{J}(\theta)$$

$$\tilde{J}(\theta) = \frac{1}{2}(\theta^T x^{(u)} - y^{(u)})^2$$

* Only linear regression has a closed form solution of GD. In other cases, the solution varies with the problem.

We defined $J(\theta)$ as,

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

Design Matrix

$$\overset{n \times d+1}{X} = \begin{pmatrix} x^{(i)} \\ \vdots \\ x^{(i+1)} \\ \vdots \\ x^{n} \end{pmatrix} \qquad \overline{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} \qquad \theta = \begin{pmatrix} \theta_1 \\ \vdots \\ \theta_{d+1} \end{pmatrix}$$

$X\theta - y$

$\mathbb{R}^{n \times d+1} \mathbb{R}^{d+1} - \mathbb{R}^n = \mathbb{R}^n$

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

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

Now, $\nabla_\theta J(\theta) = 0$

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

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

$$= \nabla_\theta \tfrac{1}{2} \left(\theta^T (X^T X) \theta - 2\theta^T (X^T y) + y^T y \right)$$

$$= \frac{1}{2} \left(2(X^T X)\theta - 2X^T y \right) = 0$$

$$\Rightarrow X^T X \theta = X^T y \quad \text{or,} \quad \hat{\theta} = (X^T X)^{-1} X^T y$$

↳ Normal Expression

{As long as X^T is invertible}

*Probabilistic Interpretation

$$y^{(i)} = \theta^T x^{(i)} + \varepsilon^{(i)}$$

$$\varepsilon \approx N(0, \sigma^2)$$