Supervised learning setup

Notation.

M: # samples

X: features (input)

Y: target (output)

(X, Y): training data

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

Functionality

Training sep

↓

Learning algo

↓

features → $\boxed{h}$ → target

hypothesis

Formalization Initially (linear regression): $h(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots +$

Define $x_0 = 1$:

$$h(x) = \sum_{j=0}^{n} \theta_j x_j \quad (n = \#\text{features})$$

Define $\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} \quad x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}$

$$\Rightarrow h(x) = \theta^T x$$

Cost function   Consider $h_\theta(x) - y$

prediction   observed true value

Traditional learning problem:

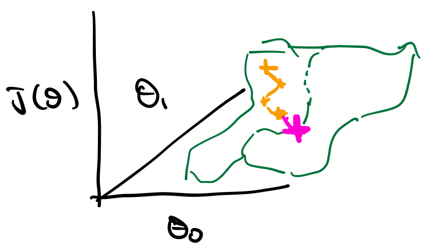
$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

$$\theta_f = \arg\min_\theta J(\theta)$$

Note h maps x to y

J maps θ to $\mathbb{R}$

Gradient Descent



Objective: find θ_i's that minimize $J(\theta)$ over (x, y)

Idea:
1. choose random starting location $(\theta = \vec{0})$
2. find direction of steepest descent
3. take step of predetermined size α in that direction
   (change θ to reduce $J(\theta)$)

   $$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \quad j = 0 \ldots n$$

   α: learning rate

   $$\boxed{\theta := \theta - \alpha \nabla J(\theta)}$$

4. if not reached local min, goto 2
5. profit!

Further derivation

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

Simplified: $J(\theta) = \frac{1}{2}\left(h_\theta(x) - y\right)^2$

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

$$= \left(h_\theta(x) - y\right)\frac{\partial}{\partial\theta_j}\left(h_\theta(x) - y\right)$$

$$= \left(h_\theta(x) - y\right)\frac{\partial}{\partial\theta_j}\left(\sum_{i=1}^{n}\theta_i x_i - y\right)$$

$$= \boxed{\left(h_\theta(x) - y\right) x_j}$$

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

$$\Rightarrow \quad \theta_j = \theta_j - \alpha\left(h_\theta(x) - y\right) x_j$$

Final definition

Repeatedly:

$$\boxed{\theta_j = \theta_j - \alpha\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right) x_j^{(i)} \qquad j = 1 \ldots n}$$

Batch Gradient Descent - GD wrt entire training data

Stochastic Gradient Descent - much faster

Repeat: {

  For $i = 1 \ldots m$ {

    $\theta_j := \theta_j - \alpha\left(h_\theta(x^{(i)}) - y^{(i)}\right) x_j^{(i)} \qquad j = 1 \ldots m$

  }

}

Minibatch Gradient Descent - look at 64 or 128 examples, do SGD step per minibatch

Explicitly solving linear regression

$$X = \begin{bmatrix} — x^{(1)T} — \\ — x^{(2)T} — \\ \vdots \\ — x^{(m)T} — \end{bmatrix} \in \mathbb{R}^{m \times n+1} \quad \text{(design matrix)}$$

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

Normal equation

$\theta^* =$ optimal value for θ

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

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