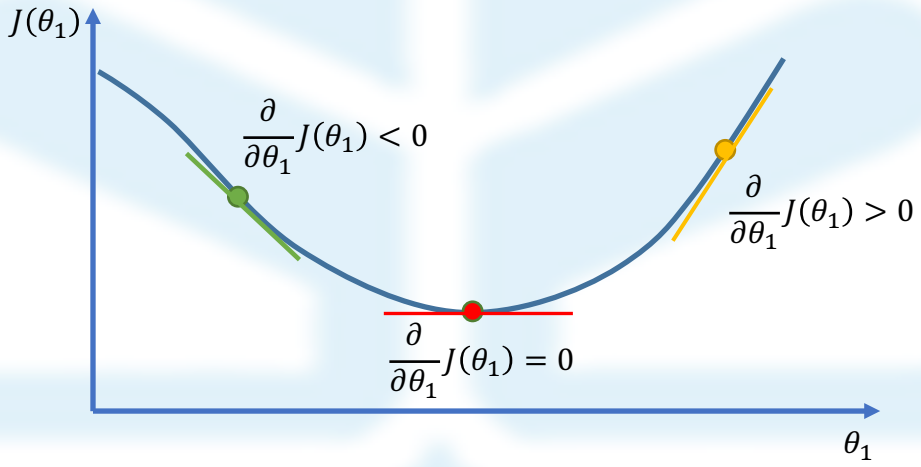
به نام او، به یاد او، برای او

Machine Learning

Linear Regression



Dr. *Ali Valinejad*

valinejad.ir
valinejad@umz.ac.ir

Outline:

- Supervised Learning
- Linear Regression
- Normal equations
- Gradient descent
  - Batch Gradient Descent
  - Stochastic Gradient Descent
  - Mini-Batch Gradient Descent
- Linear Regression: Probabilistic interpretation
- Locally weighted linear regression

Supervised Learning

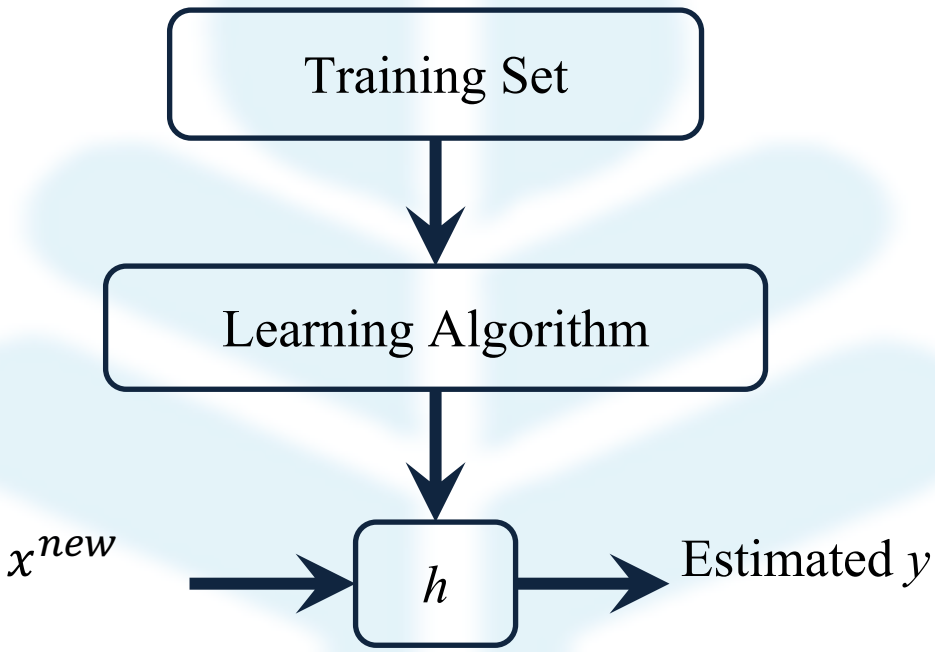
Supervised Learning

Price of a House

Size:							
Price:	50	63	75	?	102	142	198

what do you think is the best guess for the price of the house  ?

Supervised Learning



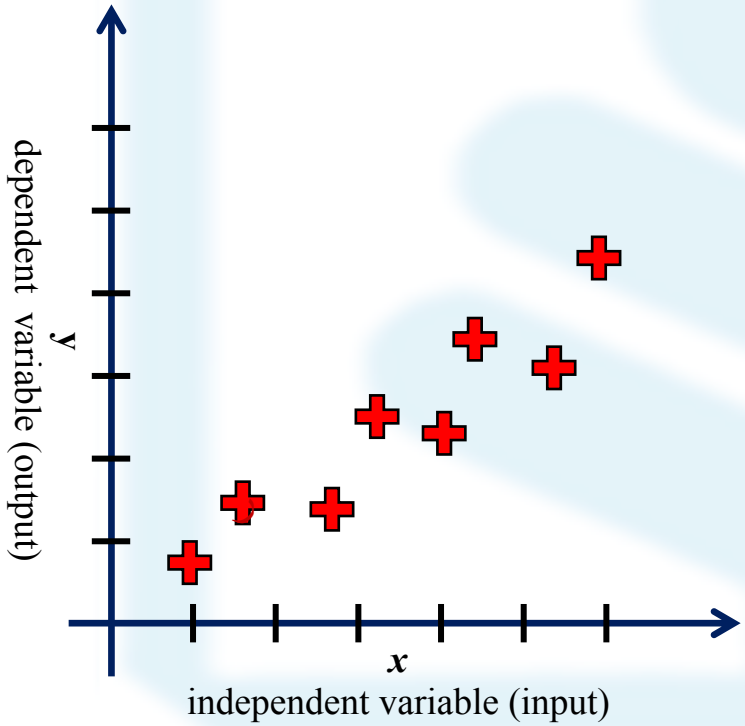
In order to design a learning algorithm, we have to answer the following question:

How we want to represent Hypothesis?

x^{new}

Linear Regression

Linear Regression (One Variable)



- **One feature:** x
- **Hypothesis:** $h_\theta(x) = \theta_0 + \theta_1 x$
- **Parameters:** θ_0, θ_1
- **Cost function:**

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

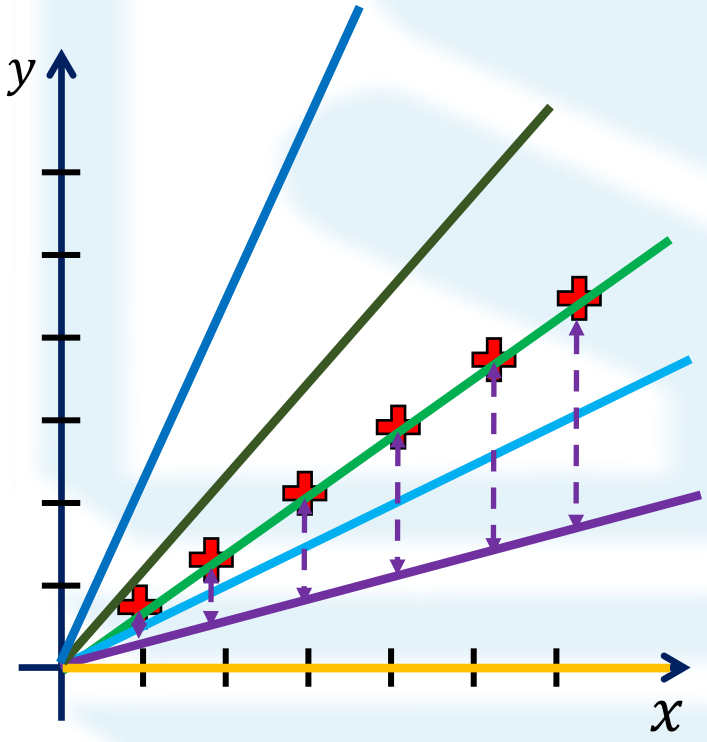
- **Goal**: minimize $J(\theta_0, \theta_1)$

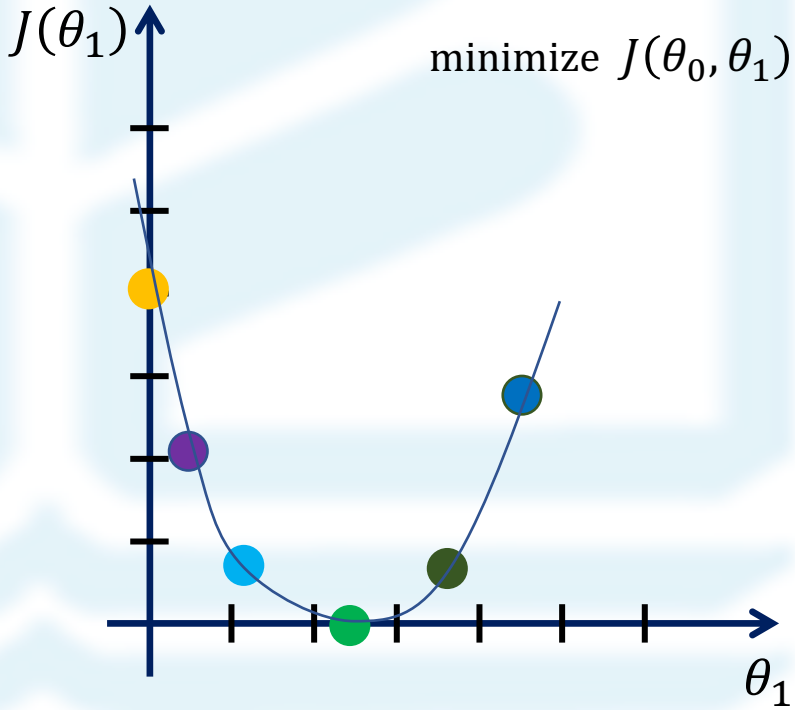
Idea: Choose θ_0, θ_1 so that $h_\theta(x)$ is close to y for our training examples (x,y).

Linear Regression (One Variable)

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

Assume $\theta_0 = 0, so\ h_\theta(x) = \theta_1 x$ and $J(\theta_0, \theta_1) = J(\theta_1)$





minimize $J(\theta_0, \theta_1)$

Linear Regression (Multiple Variables)

- **Multiple features:** $x_1, x_2, \dots, x_n$
- **Hypothesis:** $x_0 \coloneqq 1 \Rightarrow h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n = \boldsymbol{\theta}^\top x$
- **Parameters:** $\theta_0, \theta_1, \dots, \theta_n$
- **Cost function:**

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

- **Goal**: minimize $J(\boldsymbol{\theta}) \coloneqq J(\theta_0, \theta_1, \dots, \theta_n)$

- **Learning**
  - *Solving normal equation* $\boldsymbol{\theta} = (X^\top X)^{-1} X^\top y$
  - *Gradient descent*
- **Inference**

$$\hat{y} = h_\theta(x^{\text{test}}) = \boldsymbol{\theta}^\top x^{\text{test}}$$

Normal equations **to minimize $J(\theta)$**

Normal equations to minimize $J(\theta)$

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

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, J(\boldsymbol{\theta})$$

Normal equations to minimize $J(\theta)$

Given a training set

$x_1^{(i)}$	...	$x_n^{(i)}$	$y^{(i)}$
$x_1^{(1)}$	...	$x_n^{(1)}$	$y^{(1)}$
$x_1^{(2)}$	...	$x_n^{(2)}$	$y^{(2)}$
$\vdots$	$\ddots$	$\vdots$	$\vdots$
$x_1^{(m)}$	...	$x_n^{(m)}$	$y^{(m)}$

set $x_0^{(i)} = 1,\ i = 1,2,\dots,m$

$x_0^{(i)}$	$x_1^{(i)}$	...	$x_n^{(i)}$	$y^{(i)}$
1	$x_1^{(1)}$	...	$x_n^{(1)}$	$y^{(1)}$
1	$x_1^{(2)}$	...	$x_n^{(2)}$	$y^{(2)}$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
1	$x_1^{(m)}$	...	$x_n^{(m)}$	$y^{(m)}$

Normal equations to minimize $J(\theta)$

Given a training set, define the ***design matrix*** X

$$X := \begin{bmatrix} - & (x^{(1)})^T & - \\ - & (x^{(2)})^T & - \\ & \vdots & \\ - & (x^{(m)})^T & - \end{bmatrix} = \begin{bmatrix} 1 & x_1^{(1)} & \dots & x_n^{(1)} \\ 1 & x_1^{(2)} & \dots & x_n^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(m)} & \dots & x_n^{(m)} \end{bmatrix} \in \mathbb{R}^{m \times (n+1)}$$

where

$$x^{(i)} = \begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix} \in \mathbb{R}^{n+1}$$

set:

$$y := \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix} \in \mathbb{R}^m \qquad \theta := \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^{n+1}$$

Normal equations to minimize $J(\theta)$

$$x^{(i)} = \begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix} \quad X := \begin{bmatrix} 1 & x_1^{(1)} & \dots & x_n^{(1)} \\ 1 & x_1^{(2)} & \dots & x_n^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(m)} & \dots & x_n^{(m)} \end{bmatrix} \quad y := \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix} \quad \theta := \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$$

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n = \sum_{j=0}^{n} \theta_j x_j$$

$$\Rightarrow \boxed{h_\theta(x) = x^\top \theta}$$

$$\Rightarrow \boxed{h_\theta(x^{(i)}) = (x^{(i)})^T \theta}$$

$$\Rightarrow \boxed{h_\theta(x^{(i)}) - y^{(i)} = (x^{(i)})^T \theta - y^{(i)}}$$

$$\Rightarrow \boxed{X^T\theta - y = \begin{bmatrix} (x^{(1)})^T\theta - y^{(1)} \\ (x^{(2)})^T\theta - y^{(2)} \\ \vdots \\ (x^{(m)})^T\theta - y^{(m)} \end{bmatrix}}$$

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

Normal equations to minimize $J(\theta)$

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

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

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

if $\nabla_\theta J(\theta^*) = \mathbf{0}$ $\boxed{\textit{then}\quad \theta^* = (X^T X)^{-1}X^T y}$

$$\boxed{\theta^* = \underset{\theta}{\operatorname{argmin}}\, J(\theta)}$$

Normal equations to minimize $J(\theta)$

Examples: $m = 5$.

x_0	Size (feet²) x_1	Number of bedrooms x_2	Number of floors x_3	Age of home (years) x_4	Price ($1000) y
1	2104	5	1	45	460
1	1416	3	2	40	232
1	1534	3	2	30	315
1	852	2	1	36	178
1	3000	4	1	38	540

(Andrew Ng)

$$X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \\ 1 & 3000 & 4 & 1 & 38 \end{bmatrix} \qquad y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \\ 540 \end{bmatrix}$$

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

Normal equations to minimize $J(\theta)$

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

What if $X^T X$ is non-invertible?

- *Redundant features (linearly dependent).*

  e.g. if $x_1 = size(in\ feet^2)$ and $x_2 = size(in\ m^2)$ then x_1 and x_2 are linearly independent. x_1 can be removed.

- *Too many features (e.g. #training example $\leq$ #features).*

  Delete some features, or use regularization.

(Andrew Ng)

Gradient descent to minimize $J(\theta)$

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

$$\theta^* = \underset{\theta}{\operatorname{argmin}}\, J(\theta)$$

Gradient Descent

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

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

For a single training example ($m = 1$),

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

This rule is called the **LMS** (*least mean squares*) update rule, and is also known as the **Widrow-Hoff** learning rule.

Gradient descent to minimize $J(\theta)$

Three types of gradient descents

- **Batch Gradient Descent:**
  Parameters are updated after computing the gradient of error with respect to the *entire training set*
- **Stochastic Gradient Descent(SGD):**
  Parameters are updated after computing the gradient of error with respect to a *single training example*
- **Mini Batch Gradient Descent:**
  Parameters are updated after computing the gradient of error with respect to a *subset of the training set*

Gradient descent to minimize $J(\theta)$

Batch Gradient Descent

Batch Gradient Descent

Repeat {

$$\boldsymbol{\theta} := \boldsymbol{\theta} - \alpha \nabla_{\theta} J(\boldsymbol{\theta})$$

}

α: Learning rate

Batch Gradient Descent

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n = \sum_{j=0}^{n} \theta_j x_j$$

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2 \quad \Rightarrow \quad \nabla_\theta J(\theta) = \begin{bmatrix} \frac{\partial J(\theta)}{\partial \theta_0} \\ \frac{\partial J(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{bmatrix}$$

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

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)}, \qquad j = 1,2,\dots,n$$

Batch Gradient Descent

Repeat {

$$\boldsymbol{\theta} := \boldsymbol{\theta} - \alpha \nabla_{\theta} J(\boldsymbol{\theta})$$

}

α: Learning rate

Repeat {

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

(simultaneously update θ_j for every $j = 0,1, \dots, n$)

}

Batch Gradient Descent

Repeat {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)} \right)$$

$$\theta_j := \theta_j - \alpha \frac{1}{m} \left[\sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)} \right) x_j^{(i)} \right]$$

(simultaneously update for every $j = 0,1, \ldots, n$)

}

This rule is called the **Least Mean Squares** (**LMS**) update rule for a training set of ***m*** data points, which is also known as the **Widrow-Hoff learning** rule.

Batch Gradient Descent

```
def gradient_descent(X, y, theta, learning_rate, iterations):
    m, n = X.shape
    n = n-1
    cost_history = np.zeros(iterations)
    theta_history = np.zeros((iterations, n))
    for it in range(iterations):
        prediction = np.dot(X,theta)
        theta = theta -(1/m)*learning_rate*( X.T.dot((prediction - y)))
        theta_history[it,:] = theta.T
        cost_history[it]  = cal_cost(theta,X, y)
    return theta, cost_history, theta_history
# ------------------------------------------------------------------------------------------------------- #
def  cal_cost(theta, X, y):
    '''    Calculates the cost for given theta, X and Y.)   '''
    m = len(y)
    predictions = X.dot(theta)
    cost = 1/(2*m) * np.sum(np.square(predictions-y))
    return cost
```


Gradient descent to minimize $J(\theta)$

Stochastic Gradient Descent(SGD)

Stochastic Gradient Descent

Randomly shuffle(reorder) examples in training set

Repeat {

for *i*=1 to *m* {

$$\theta_j := \theta_j - \alpha \frac{1}{m}\left[\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)}\right]$$

(simultaneously update for every $j = 0,1, \ldots, n$)

}

}

Stochastic Gradient Descent

Randomly shuffle(reorder) examples in training set

Repeat {

for *i*=1 to *m* {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)$$

$$\theta_j := \theta_j - \alpha \frac{1}{m}\left[\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)}\right]$$

(simultaneously update for every $j = 0,1,\dots,n$)

}

}

Stochastic Gradient Descent

```
def stocashtic_gradient_descent(X, y, theta, learning_rate, iterations):
    m = len(y)
    cost_history = np.zeros(iterations)
    for it in range(iterations):
        cost = 0.0
        for i in range(m):
            rand_ind = np.random.randint(0, m)
            X_i = X[rand_ind,:].reshape(1, X.shape[1])
            y_i = y[rand_ind].reshape(1, 1)
            prediction = np.dot(X_i, theta)
            theta = theta -(1/m)*learning_rate*( X_i.T.dot((prediction - y_i)))
            cost += cal_cost(theta, X_i, y_i)
        cost_history[it]  = cost
    return theta, cost_history
# ------------------------------------------------------------------------------------------------------- #
def  cal_cost(theta, X, y):
    '''    Calculates the cost for given theta, X and Y.)   '''
    m = len(y)
    predictions = X.dot(theta)
    cost = 1/(2*m) * np.sum(np.square(predictions-y))
    return cost
```

Gradient descent to minimize $J(\theta)$

Mini Batch Gradient Descent(SGD)

Mini-Batch Gradient Descent

Given training set:

$x^{(1)}$	$x^{(2)}$		…		$x^{(i-1)}$	$x^{(i)}$	$x^{(i+1)}$			…		$x^{(m)}$
$y^{(1)}$	$y^{(2)}$		…		$y^{(i-1)}$	$y^{(i)}$	$y^{(i+1)}$			…		$y^{(m)}$

first mini-batch second mini-batch s^{th} mini-batch

$x^{(1)}$	$x^{(2)}$	…	$x^{(b)}$	$x^{(b+1)}$	$x^{(b+2)}$	…	$x^{(2b)}$	…				$x^{(m)}$
$y^{(1)}$	$y^{(2)}$	…	$y^{(b)}$	$y^{(b+1)}$	$y^{(b+2)}$	…	$y^{(2b)}$	…				$y^{(m)}$

training example in each mini-batch: b

mini-batch: $s = \left\lceil \frac{m}{b} \right\rceil$

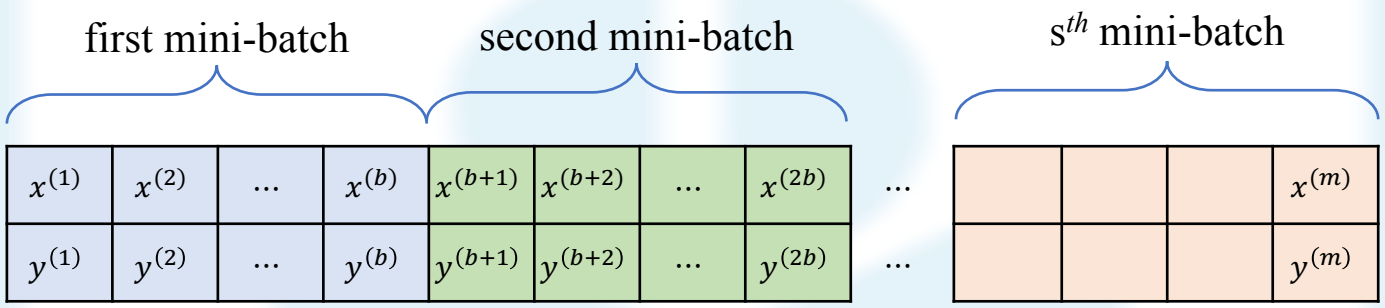
first mini-batch: $x^{(1)}, x^{(2)}, \dots, x^{(b)}$

second mini-batch: $x^{(b+1)}, x^{(b+2)}, \dots, x^{(2b)}$

$\vdots$

k^{th} mini-batch: $x^{((k-1)b+1)}, x^{((k-1)b+2)}, \dots, x^{(kb)}$

Mini-Batch Gradient Descent



Repeat {

Randomly shuffle(reorder) examples in training set

partition new training set into $s = \left[\frac{m}{b}\right]$ *mini-batches of size* b

for k=1 to s {

$$set \quad X^{\{k\}} = \left\{x^{((k-1)b+1)}, x^{((k-1)b+2)}, \dots, x^{(kb)}\right\}$$

$$set \quad Y^{\{k\}} = \left\{y^{((k-1)b+1)}, y^{((k-1)b+2)}, \dots, y^{(kb)}\right\}$$

$$\theta_0 := \theta_0 - \alpha \frac{1}{b} \sum_{i=1}^{b} \left(h_\theta\left(X^{\{k\}}(i)\right) - Y^{\{k\}}(i)\right)$$

$$\theta_j := \theta_0 - \alpha \frac{1}{b} \sum_{i=1}^{b} \left(h_\theta\left(X^{\{k\}}(i)\right) - Y^{\{k\}}(i)\right) X^{\{k\}}(i,j)$$

(simultaneously update for every $j = 0,1, \dots, n$)

}

}

$$X^{\{k\}}(i,j) := x_j^{((k-1)b+i}$$
$$Y^{\{k\}}(i) := y^{((k-1)b+i)}$$

Mini-Batch Gradient Descent

```
def mini_batch_gradient_descent(X, y, theta, learning_rate, iterations, batch_size):
    m = len(y)
    cost_history = np.zeros(iterations)
    n_batches = int(m/batch_size)
    for it in range(iterations):
        cost =0.0
        indices = np.random.permutation(m)
        X = X[indices]
        y = y[indices]
        for i in range(0, m, batch_size):
            X_i = X[i:i+batch_size]
            Y_i = y[i:i+batch_size]
            X_i = np.c_[np.ones(len(X_i)), X_i]
            prediction = np.dot(X_i,theta)
            theta = theta -(1/m)*learning_rate*( X_i.T.dot((prediction - Y_i)))
            cost += cal_cost(theta,X_i,Y_i)
        cost_history[it]  = cost
    return theta, cost_history
# ---------------------------------------------------------------------------------------------- #
def  cal_cost(theta, X, y):
    '''    Calculates the cost for given theta, X and Y.)   '''
    m = len(y)
    predictions = X.dot(theta)
    cost = 1/(2*m) * np.sum(np.square(predictions-y))
    return cost
```

Gradient Descent

Batch Gradient Descent	Stochastic Gradient Descent	Mini-Batch Gradient Descent
Since entire training data is considered before taking a step in the direction of gradient, therefore it takes a lot of time for making a single update.	Since only a single training example is considered before taking a step in the direction of gradient, we are forced to loop over the training set and thus cannot exploit the speed associated with vectorizing the code.	Since a subset of training examples is considered, it can make quick updates in the model parameters and can also exploit the speed associated with vectorizing the code.
It makes smooth updates in the model parameters	It makes *very noisy* updates in the parameters	Depending upon the batch size, the updates can be made *less noisy* – greater the batch size less noisy is the update

Thus, mini-batch gradient descent makes a compromise between the speedy convergence and the noise associated with gradient update which makes it a more flexible and robust algorithm.

Linear Regression

Probabilistic interpretation

Linear Regression: Probabilistic interpretation

Assumptions:

1- There is a linear relationship between target variables $y^{(i)}$ and the inputs $x^{(i)}$ via the equation $y^{(i)} = \theta_0 + \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} + \cdots + \theta_n x_n^{(i)} + \epsilon^{(i)} = \boldsymbol{\theta}^T x^{(i)} + \epsilon^{(i)}$

2- For i =1, 2, …, m, $\epsilon^{(i)}$ are distributed **iid** (independently and identically distributed) according to a Gaussian distribution with *mean zero* and some *variance* σ^2, i.e. $\epsilon^{(i)} \sim \mathcal{N}(0, \sigma^2)$

$\epsilon^{(i)}$ is an error term that captures either unmodeled effects (such as if *there are some features very pertinent to predicting*, *but that we'd left out of the regression*), or *random noise*.

Based on above probabilistic assumption in a regression problem,

- why might ***linear regression***, be a reasonable choice?
- why might the ***least-squares cost function*** $J(\boldsymbol{\theta})$, be a reasonable choice?

Linear Regression: Probabilistic interpretation

$\epsilon^{(i)} \sim \mathcal{N}(0, \sigma^2)$ implies that the *density* of $\epsilon^{(i)}$ is given by

$$p\left(\epsilon^{(i)}\right) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{\left(\epsilon^{(i)}\right)^2}{2\sigma^2}}$$

$$\mathbb{E}\left[\epsilon^{(i)}\right] = 0 \quad \Rightarrow \quad \mathbb{E}\left[y^{(i)} - \boldsymbol{\theta}^T x^{(i)}\right] = 0$$

$$\text{Var}\left[\epsilon^{(i)}\right] = \sigma^2 \Rightarrow \text{Var}\left[y^{(i)} - \boldsymbol{\theta}^T x^{(i)}\right] = \sigma^2$$

$$p\left(y^{(i)} - \boldsymbol{\theta}^T x^{(i)}\right) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{\left(y^{(i)} - \boldsymbol{\theta}^T x^{(i)}\right)^2}{2\sigma^2}}$$

Linear Regression: Probabilistic interpretation

$\epsilon^{(i)} \sim \mathcal{N}(0, \sigma^2)$ implies that the *density* of $\epsilon^{(i)}$ is given by

$$p\left(\epsilon^{(i)}\right) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{\left(\epsilon^{(i)}\right)^2}{2\sigma^2}}$$

Since $\boldsymbol{\theta}^T x^{(i)}$ is *constant* given $x^{(i)}$ and $\epsilon^{(i)}$ has a mean of zero, $y^{(i)} = \boldsymbol{\theta}^T x^{(i)} + \epsilon^{(i)}$ implies that

$$\mathbb{E}\left[y^{(i)} | x^{(i)}; \boldsymbol{\theta}\right] = \mathbb{E}\left[\boldsymbol{\theta}^T x^{(i)} + \epsilon^{(i)} | x^{(i)}; \boldsymbol{\theta}\right] = \boldsymbol{\theta}^T x^{(i)} + \mathbb{E}\left[\epsilon^{(i)} | x^{(i)}; \boldsymbol{\theta}\right] = \boldsymbol{\theta}^T x^{(i)}$$

$$\text{Var}\left[y^{(i)} | x^{(i)}; \boldsymbol{\theta}\right] = \text{Var}\left[\boldsymbol{\theta}^T x^{(i)} + \epsilon^{(i)} | x^{(i)}; \boldsymbol{\theta}\right] = \text{Var}\left[\epsilon^{(i)} | x^{(i)}; \boldsymbol{\theta}\right] = \sigma^2$$

Since $\boldsymbol{\theta}^T x^{(i)}$ is *constant* given $x^{(i)}$ and $\epsilon^{(i)}$ has a mean of zero.
Similarly, $y^{(i)}$ does not necessarily have a normal distribution in this type of linear regression model, but the assumptions imply that the conditional distribution of $y^{(i)}$ given $x^{(i)}$ is normally distributed with mean $\boldsymbol{\theta}^T x^{(i)}$ and standard deviation σ.

$$p\left(y^{(i)} | x^{(i)}; \boldsymbol{\theta}\right) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{\left(y^{(i)} - \boldsymbol{\theta}^T x^{(i)}\right)^2}{2\sigma^2}}$$

So, We can write the distribution of $y^{(i)}$ as

$$y^{(i)} | x^{(i)}; \theta \sim \mathcal{N}(\boldsymbol{\theta}^T x^{(i)}, \sigma^2)$$

Linear Regression: Probabilistic interpretation

$$p\left(y^{(i)}|x^{(i)};\boldsymbol{\theta}\right)=\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{\left(y^{(i)}-\boldsymbol{\theta}^{T}x^{(i)}\right)^{2}}{2\sigma^{2}}}$$

Given X (the *design matrix*, which contains all the $x^{(i)}$'s) and $\boldsymbol{\theta}$, *what is the distribution of the* $\boldsymbol{y}=[y^{(1)}, y^{(2)}, \ldots, y^{(m)}]^{T}$*?*

- Given any data set $(x^{(1)}, y^{(1)})$, $(x^{(2)}, y^{(2)})$, …, $(x^{(m)}, y^{(m)})$, the probability of the data is given by $p(\boldsymbol{y}|X;\boldsymbol{\theta})$. This quantity is typically viewed as a function of $\boldsymbol{y}$ (and perhaps X), for a fixed value of $\boldsymbol{\theta}$.

- When we wish to explicitly view this as a function of $\boldsymbol{\theta}$, we will instead call it the ***likelihood function***:

$$L(\boldsymbol{\theta})=L(\boldsymbol{\theta};X,\boldsymbol{y})=p(\boldsymbol{y}|X;\boldsymbol{\theta})$$

by the independence assumption on the $\epsilon^{(i)}$'s (and hence also the $y^{(i)}$'s given the $x^{(i)}$'s), this can also be written

$$L(\boldsymbol{\theta})=\prod_{i=1}^{m}p\left(y^{(i)}|x^{(i)};\boldsymbol{\theta}\right)$$

$$=\prod_{i=1}^{m}\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{\left(y^{(i)}-\boldsymbol{\theta}^{T}x^{(i)}\right)^{2}}{2\sigma^{2}}}$$

Linear Regression: Probabilistic interpretation

Maximum Likelihood

The principal of maximum likelihood says that we should choose $\boldsymbol{\theta}$ so as to make the data as high probability as possible. i.e., we should choose $\boldsymbol{\theta}$ to maximize $L(\boldsymbol{\theta})$.

$$\theta^* = \underset{\theta}{\operatorname{argmax}}\, L(\theta)$$

Instead of maximizing $L(\boldsymbol{\theta})$, we can also maximize any strictly increasing function of $L(\boldsymbol{\theta})$

$$l(\theta) := \log L(\theta)$$

$$\theta^* = \underset{\theta}{\operatorname{argmax}}\, l(\theta)$$

Linear Regression: Probabilistic interpretation

$$l(\boldsymbol{\theta}) := \log L(\boldsymbol{\theta})$$

$$= \log \prod_{i=1}^{m} p\left(y^{(i)} | x^{(i)}; \boldsymbol{\theta}\right)$$

$$= \sum_{i=1}^{m} \log \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{\left(y^{(i)} - \boldsymbol{\theta}^T x^{(i)}\right)^2}{2\sigma^2}}$$

$$= m \log \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{2\sigma^2} \sum_{i=1}^{m} \left(y^{(i)} - \boldsymbol{\theta}^T x^{(i)}\right)^2$$

$$\Rightarrow \quad \max_{\theta} l(\theta) \ \textit{is equivalent to} \ \min_{\theta} \frac{1}{2} \sum_{i=1}^{m} \left(y^{(i)} - \boldsymbol{\theta}^T x^{(i)}\right)^2$$

$$\Rightarrow \quad \max_{\theta} l(\theta) \ \textit{is equivalent to} \ \min_{\theta} J(\boldsymbol{\theta})$$

- Under the previous probabilistic assumptions on the data, least-squares regression corresponds to finding the maximum likelihood estimate of $\boldsymbol{\theta}$.

Linear Regression

Locally weighted linear regression

Locally weighted linear regression

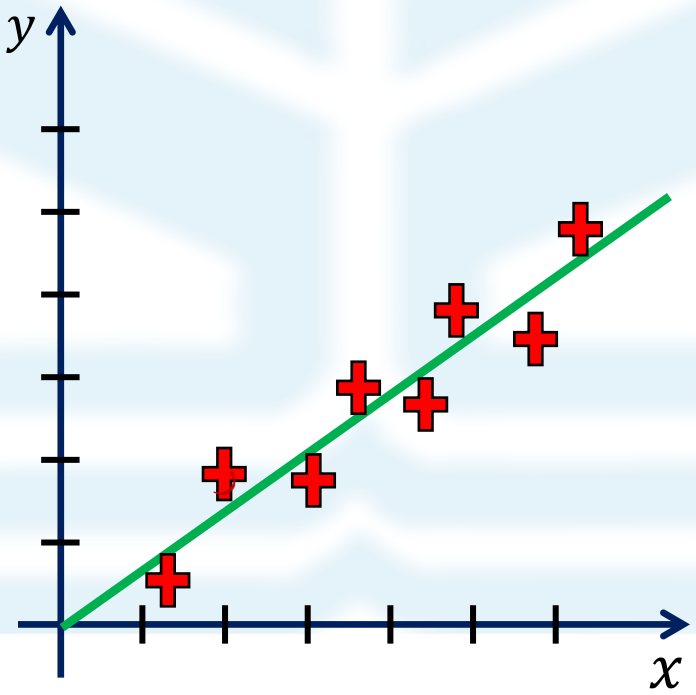
Training phase:

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

Predict output:

return $x^\top\theta$

linear regression on linear data



Locally weighted linear regression

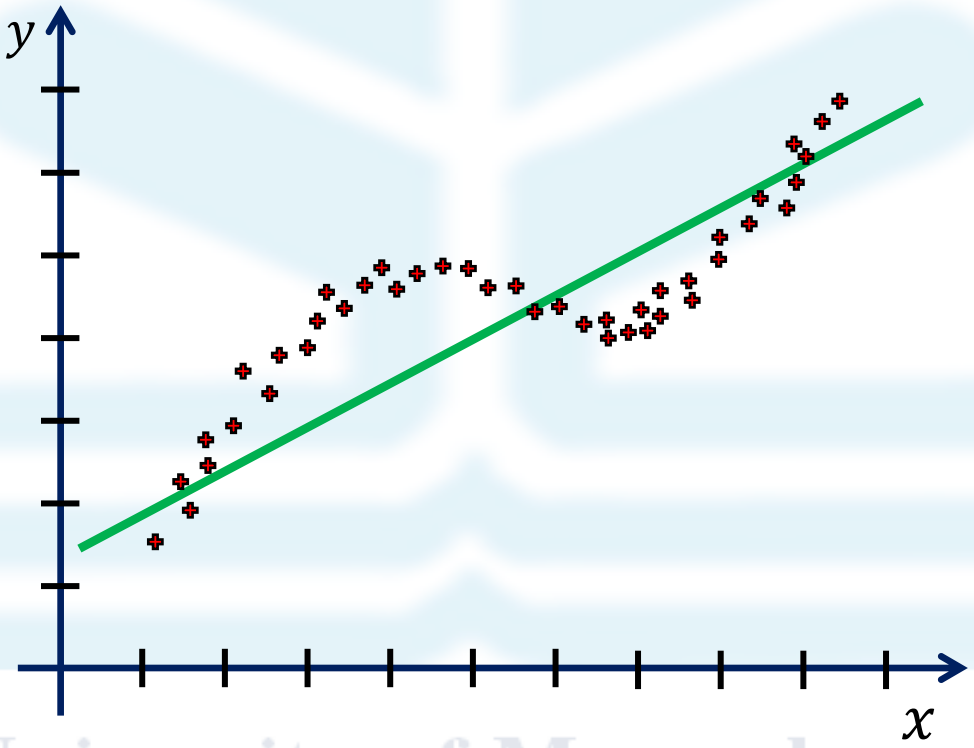
Training phase:

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

Predict output:

return $x^\top\theta$

linear regression on non-linear data



Locally weighted linear regression

- Locally weighted linear regression is ***a non-parametric algorithm***, that is, the model does not learn a fixed set of parameters as is done in ordinary linear regression.

- Parameters θ are computed *individually* for each ***query point*** x.

- While computing θ, a ***higher preference*** is given to the points in the training set lying in the ***vicinity*** of x than the points lying far away from x.

The modified cost function is:

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

Locally weighted linear regression

The modified cost function is:

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

- where, $\omega^{(i)}$ is a non-negative "***weight***" associated with training point $x^{(i)}$.
- For $x^{(i)}$ s lying closer to the query point x, the value of $\omega^{(i)}$ is ***large***,
- For $x^{(i)}$ s lying far away from x the value of $\omega^{(i)}$ is ***small***.

Thus, the training-set-points lying ***closer*** to the query point x ***contribute*** more to the cost $J(\theta)$ than the points lying far away from x.

A typical choice of $\omega^{(i)}$ is:

$$\omega^{(i)} = e^{-\frac{\left(x^{(i)}-x\right)^2}{2\tau^2}}$$

- where, τ is called the ***bandwidth parameter*** and controls the *rate* at which $\omega^{(i)}$ falls with distance from x

Locally weighted linear regression

Training phase:

Compute θ to minimize $J(\theta) = \frac{1}{2m}\sum_{i=1}^{m} \boldsymbol{\omega^{(i)}}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$

Predict output:

return $x^{\top}\theta$

Points to remember:

- Locally weighted linear regression is a supervised learning algorithm.
- It is a non-parametric algorithm.
- There exists No training phase.
- All the work is done during the testing phase/while making predictions.

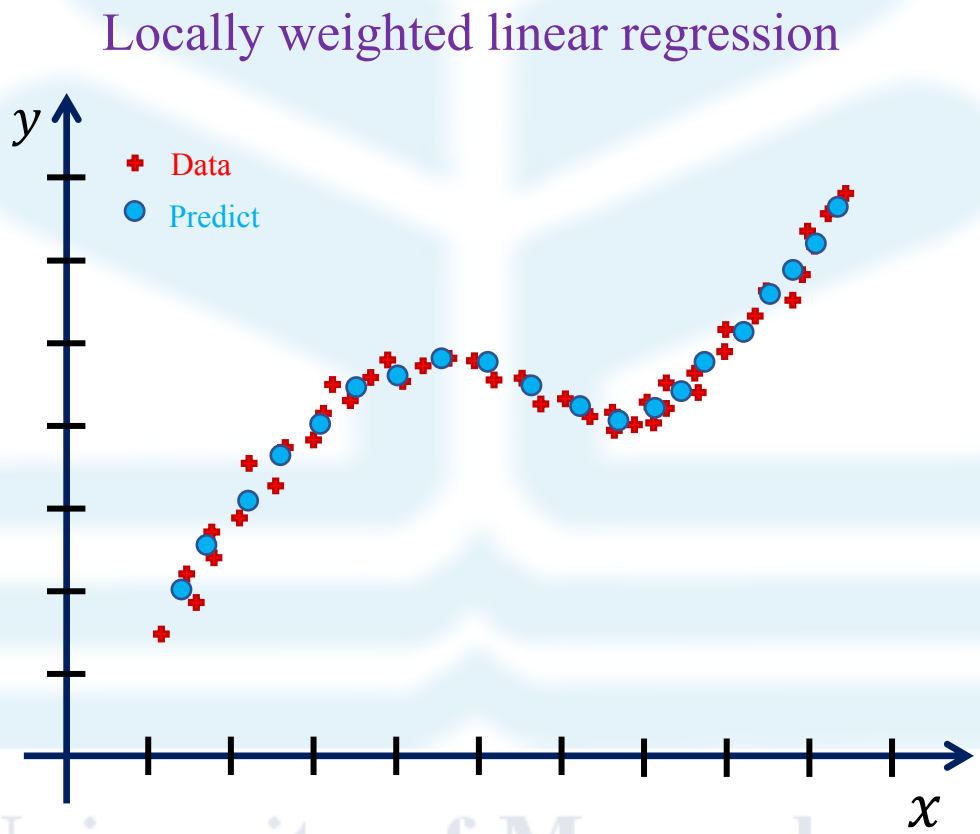
Locally weighted linear regression

Training phase:

Compute θ to minimize $J(\theta) = \frac{1}{2m}\sum_{i=1}^{m} \boldsymbol{\omega^{(i)}}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$

Predict output:

return $x^\top\theta$



References

https://www.geeksforgeeks.org/gradient-descent-algorithm-and-its-variants/

https://www.geeksforgeeks.org/ml-mini-batch-gradient-descent-with-python/

https://www.holehouse.org/mlclass/04_Linear_Regression_with_multiple_variables.html

Andrew Ng, https://www.coursera.org/learn/machine-learning