

§2. SUPERVISED LEARNING SETUP & LINEAR REGRESSION

06.04

OUTLINE

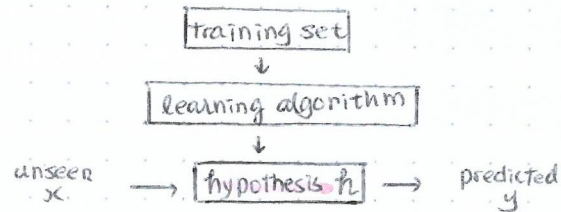
- ① choose hypothesis ② choose cost fx
- linear regression + ℓ_2 loss (= OLS). } 3 fundamental building blocks of a supervised learning algorithm.
- Batch/stochastic gradient descent.
- Normal equation. ③ parameter learning/optimization. (parametric)

SUPERVISED LEARNING / REGRESSION

{Ex} housing price prediction:

	x_1	x_2	x_j	y
	size (ft ²)	# bedrooms	...	price (\$1000s)
$(x^{(1)}, y^{(1)})$	2604	3	...	400
$(x^{(2)}, y^{(2)})$	1600	3	...	330
$(x^{(3)}, y^{(3)})$	2400	3	...	369
\vdots	\vdots	\vdots	\vdots	\vdots

Process of supervised learning:



Notation:

- x : input variable / features
- y : output variable / target variable
- (x, y) : 1 training example
- $(x^{(i)}, y^{(i)})$: i -th training example
- m : # training examples
- $\{(x^{(i)}, y^{(i)}) : i=1, \dots, m\}$: training set
- x_j : j -th feature
- n : # features

$x^{(i)}$ ↗ index into Training examples, $i=1, \dots, m$.
 j ↘ index into Features, $j=(0), 1, \dots, n$.

(eg) $x^{(1)} = 2704$
 $x_1^{(2)} = 1600$

LINEAR REGRESSION

— still one of the most widely used algorithms.

How to represent h ? ⇒ one of the simplest models: linear regression.

$h_\theta(x) = \theta_0 + \theta_1 x_1 + \dots + \theta_n x_n$, θ_j : parameter. → Technically: Affine function

→ shorthand: $h(x)$.

(linear function + intercept / translation).

To simplify notation, define dummy feature $x_0 \equiv 1$.

$$\Rightarrow h_\theta(x) = \sum_{j=0}^n \theta_j x_j = \theta^T x, \text{ where } \theta, x \in \mathbb{R}^{n+1} \text{ (zero-indexed)}.$$

Notes:

- linear regression ⇒ linear in parameters.
(eg) $h(x) = \theta_0 + \theta_1 x + \theta_2 x^2 \Rightarrow$ linear model (LV).
- (Univariate) linear regression: $\theta \in \text{scalar}$ / Simple linear regression: $x \in \text{scalar}$.
↳ Multiple linear regression: $x \in \text{vector}$.
- Multivariate linear regression: $\theta \in \text{vector}$.

SQUARED ERROR COST FUNCTION

Best E_{in} ≠ best E_{out} .

How to choose "good" θ values? ⇒ As a first attempt: choose θ s.t. $h(x)$ xy for training examples.

Define quantitative measure of model performance:

- loss function: how bad $h(x)$ is doing on 1 example.

⇒ Choose l_2 loss:

$$l(h_\theta(x), y) = \frac{1}{2} (h_\theta(x) - y)^2.$$

↳ Convention
(derivatives of l_2).

- Cost function: how bad $h_\theta(x)$ is doing on m examples.

⇒ Squared error cost function:

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

Notes:

↳ is including normalization factor $\frac{1}{m} \Rightarrow$ Mean squared error (MSE).
(same optimum).

- Linear regression + MSE = ordinary least squares (OLS).

- Why l_2 ? ⇒ There are other forms of loss (eg. l_1 loss $|h_\theta(x) - y|$), but l_2 is a common choice for regression problems:

→ works well.

→ math convenience (continuously differentiable).

→ follows naturally from $y^{(i)} \sim$ i.i.d. Gaussian noise + MLE (special case of GLMs).

GRADIENT DESCENT

How to minimize $J(\theta)$? ⇒ A common search algorithm for a differentiable objective function is gradient descent.

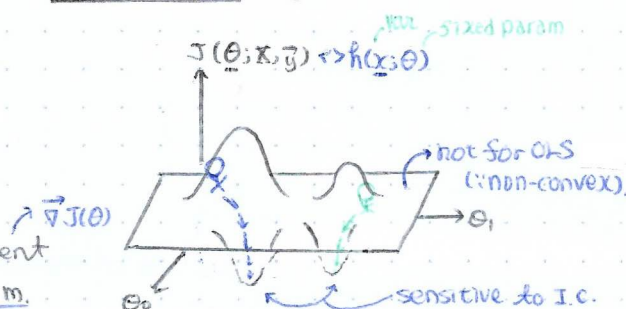
Idea:

- Start w. some initial θ (eg. zero/random initialization).

- keep changing θ at the direction of the steepest descent to reduce $J(\theta)$, until we hopefully end up at a minimum.

↳ may not converge.

↳ may converge to local min/max, saddle point.



Algorithm:

repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{\partial J(\theta)}{\partial \theta_j}, \text{ for } j=0, \dots, 1.$$

↳ simultaneous update.

$$\text{OR } \theta := \theta - \alpha \vec{\nabla}_{\theta} J(\theta).$$

assignment ($a := a+1, v$)

↳ truth assertion ($a = a+1, x$).

learning rate $\in \mathbb{R}^+$

For OLS:

Consider 1 example (linearity of differentiation):

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

$$= (h_\theta(x) - y) \cdot \frac{\partial}{\partial \theta_j} (\theta_0 x_0 + \dots + \theta_j x_j + \dots + \theta_n x_n - y)$$

$$= (h_\theta(x) - y) \cdot x_j.$$

↳ OLS + GD.

⇒ Least mean squares (LMS) update rule:

repeat until convergence {

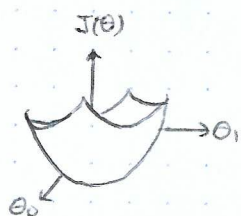
$$\theta_j := \theta_j - \alpha \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}, \forall j.$$

}

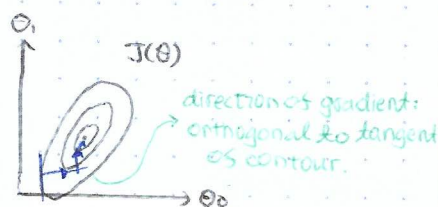
$$\theta := \theta - \frac{\alpha}{m} X^T (\vec{h} - \vec{y}).$$

(n,1) (n,m) (m,1)

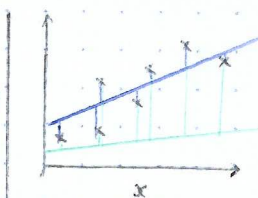
For OLS, $J(\theta)$ is a convex / quadratic \Rightarrow 1 global min.



3D surface: bowl-shaped



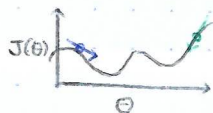
Contours: ellipses.



$\min J(\theta)$: find optimal θ that leads to the smallest sum of (vertical distance)².

Properties:

- Minus sign:



Slope $< 0 \Rightarrow$ move right.

Slope $> 0 \Rightarrow$ move left.

For maximization \Rightarrow Gradient ascent:

$$\theta := \theta + \alpha \nabla J(\theta).$$

- Local search:



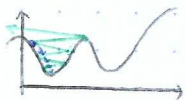
Depending on initialization, may end up at different local minimum (or wherever $\nabla J(\theta) = 0$).

- Adaptive stepsize:



When approaching min, $|\nabla J(\theta)| \downarrow$, automatically take smaller steps.
 \Rightarrow GD can converge w. a fixed α .

- Learning rate:



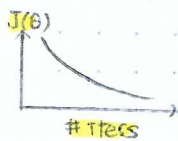
α too small: take babysteps \Rightarrow slow convergence.

α too large: overshoot minimum \Rightarrow may converge slowly, fail to converge, or even diverge.

Convergence tests

(1) Check if $\theta^{(k+1)}$ differs significantly from $\theta^{(k)}$.

(2) Monitor $J(\theta)$ as a fx of # Iters: \leftarrow much more common.



- Try a few α values on exponential scale, see which value drives $J(\theta)$ down faster.

- \therefore For sufficiently small α , $J(\theta)$ decreases on every iteration (monotonic convergence)
 \Rightarrow If $J(\theta)$ ever increases, use a smaller α .

BGD vs SGD

The algorithm introduced above is called "batch gradient descent (BGD)".

\therefore It scans through the entire Training set to take a single step. \Rightarrow slow for big data.

Alternative: Stochastic / incremental gradient descent (SGD).

repeat until convergence {

For $i = 1, \dots, m$ {

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

cost/loss for 1 example:

$$(eg) J(\theta) = \frac{1}{2} (\theta_0 x^{(i)} - y^{(i)})^2$$

}

(+) Computationally cheaper per step in comparison to BGD. \rightarrow but also lose the benefit of vectorization.

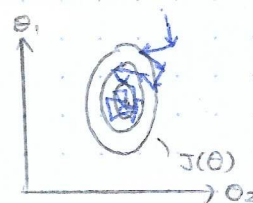
⇒ When m is large, SGD is used much often than BGD in practice.

→ $> 10^3, 10^3$

↳ Every step optimize θ to fit a particular training example

⇒ **noisy trajectory** moves towards true minimum on average,

but oscillates around it & is never converged. → the may be "close enough".



To mitigate the problem ⇒ reduce α over iterations. (**cooling schedule**)

s.t. the amplitude of oscillations ↓. → (-) one more param to worry about.

Alternatively, one can use an intermediate algorithm "mini-batch GD".

NORMAL EQUATION

Normal eqns. for nonlinear least squares ⇒ iterative solution

while GD is a general, iterative search algorithm,

in the case of OLS, it's possible to get the optimal θ in one step using a closed-form, analytic solution.

↳ In reality, computation of inverse matrix

often involves an iterative algorithm

(eg. eigendecomposition, Cholesky algorithm)

Notation for matrix derivatives

→ to simplify derivation.

Gradient:

scalar-by-vector derivative.

$$\nabla_{\theta} J(\theta) = \begin{bmatrix} \frac{\partial J}{\partial \theta_0} \\ \vdots \\ \frac{\partial J}{\partial \theta_n} \end{bmatrix} \in \mathbb{R}^{n+1}$$

Scalar-by-matrix derivative:

generalization of gradient (but no special name).

$$\nabla_A f(A) = \begin{bmatrix} \frac{\partial f}{\partial A_{11}} & \frac{\partial f}{\partial A_{12}} \\ \vdots & \vdots \\ \frac{\partial f}{\partial A_{m1}} & \frac{\partial f}{\partial A_{mn}} \end{bmatrix} \in \mathbb{R}^{m \times n}$$

↳ collection of elementwise partial derivatives (same shape).

$$(eg) A = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}, f(A) = A_{11} + A_{12}^2$$

$$\Rightarrow \nabla_A f(A) = \begin{bmatrix} 1 & 2A_{12} \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 12 \\ 0 & 0 \end{bmatrix}$$

Tr & properties

If $A \in \mathbb{R}^{n \times n}$ (square matrix), $\text{Tr}(A) = \sum_{i=1}^n A_{ii}$.

① $\text{Tr} A = \text{Tr} A^T$. → "flipping A doesn't change the diagonal elements."

② $\nabla_A \text{Tr} AB = B^T$. → (cs) $\frac{d(ab)}{da} = b$ (+ match dim),
 $f(A); B: \text{fixed param.}$

③ Invariant under cyclic permutations: →

$$\text{Tr} AB = \text{Tr} BA$$

$(m,n)(n,m) \quad (n,m)(m,n)$

$$\text{Tr} ABC = \text{Tr} BCA = \text{Tr} CAB$$

④ $\nabla_A \text{Tr} AA^T C = CA + C^T A$.

$f(A) \quad (m,m)(m,m) \quad (m,m)(m,n) \rightarrow \checkmark$

$(m,n)(n,m)(m,m)$

→ (cs) $\frac{d(a^2c)}{da} = 2ac$.

$$B = \begin{bmatrix} 1 & 1 & 1 \\ b_1 & b_2 & b_3 \end{bmatrix} \quad (2,3)$$

$$A = \begin{bmatrix} -a_1^T \\ -a_2^T \\ -a_3^T \end{bmatrix} \quad (3,2)$$

$$= b_1 a_1^T + b_2 a_2^T + b_3 a_3^T =$$

distribute element multiplication in different ways ⇒ same Tr.

$$b_{11}a_{11} + b_{21}a_{21} + b_{31}a_{31}$$

$$b_{12}a_{12} + b_{22}a_{22} + b_{32}a_{32}$$

Proof

• step 1: Express $J(\theta)$ in array notation.

Define:

"design matrix" $X = \begin{bmatrix} -x^{(1)T} \\ \vdots \\ -x^{(m)T} \end{bmatrix} \in \mathbb{R}^{m \times (n+1)}$ \neq $\vec{y} = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix} \in \mathbb{R}^m$

$$\Rightarrow X\theta = \begin{bmatrix} -x^{(1)T} \\ \vdots \\ -x^{(m)T} \end{bmatrix} \theta = \begin{bmatrix} x^{(1)T}\theta \\ \vdots \\ x^{(m)T}\theta \end{bmatrix} = \begin{bmatrix} h_{\theta}(x^{(1)}) \\ \vdots \\ h_{\theta}(x^{(m)}) \end{bmatrix} \in \mathbb{R}^m$$

For scalar s : $s^T = s$.

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

For vector v : $\|v\|^2 = \sum_i v_i^2 = v^T v$.

$$= \frac{1}{2} (X\theta - \vec{y})^T (X\theta - \vec{y}) \in \mathbb{R}.$$

• Step 2: Compute $\nabla_{\theta} J(\theta)$ using Tr properties & set to zero.

$\hookrightarrow J(\theta)$ is convex \Rightarrow global min.

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

$$= \frac{1}{2} \nabla_{\theta} (\theta^T X^T X \theta - \vec{y}^T X \theta - \theta^T X^T \vec{y} + \vec{y}^T \vec{y}) \rightarrow 0 \text{ (independent of } \theta \text{)}.$$

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

$$= \frac{1}{2} \nabla_{\theta} \underbrace{\text{Tr}(\theta^T X^T X \theta)}_{\text{a}} - \frac{1}{2} \nabla_{\theta} \underbrace{\text{Tr}(\vec{y}^T X \theta)}_{\text{b}} - \frac{1}{2} \nabla_{\theta} \underbrace{\text{Tr}(\theta^T X^T \vec{y})}_{\text{c}}$$

For (a):

$$\begin{aligned} & \nabla_{\theta} \text{Tr}(\theta^T X^T X \theta) \quad \textcircled{3} \\ &= \nabla_{\theta} \text{Tr}(\underbrace{\theta \theta^T}_{A} \underbrace{X^T X}_{C}) \quad \textcircled{4} \\ &= X^T X \theta + X^T X \theta \\ &= 2 X^T X \theta. \end{aligned}$$

For (b):

$$\begin{aligned} & \nabla_{\theta} \text{Tr}(\vec{y}^T X \theta) \quad \textcircled{3} \\ &= \nabla_{\theta} \text{Tr}(\underbrace{\vec{y}^T X}_{A} \underbrace{\theta}_{B}) \quad \textcircled{2} \\ &= X^T \vec{y}. \end{aligned}$$

For (c):

$$\begin{aligned} & \nabla_{\theta} \text{Tr}(\theta^T X^T \vec{y}) \\ &= \nabla_{\theta} \text{Tr}(\vec{y}^T X \theta) \quad (\text{Tr})s = (\text{Tr})s^T \text{ or } \textcircled{1} \\ &= \textcircled{b}. \end{aligned}$$

$$\Rightarrow \nabla_{\theta} J(\theta) = \boxed{X^T X \theta - X^T \vec{y}} \stackrel{!}{=} \vec{0} \quad \text{"Normal equation"}$$

$$\Rightarrow \theta = (X^T X)^{-1} X^T \vec{y}.$$

$$(n+1, n+1) \quad (n+1, m) \quad (m, 1) \rightarrow (n+1, 1), \forall.$$

Notes

- \therefore Normal eqn involves solving $(X^T X)^{-1} \sim \mathcal{O}(n^3)$.

\Rightarrow For large n , GD is preferred.

- When $(X^T X)$ is non-invertible:

① redundant features (linearly dependent).

② n is too large for given m (not enough linearly independent examples $x^{(i)}$'s).

\Rightarrow Solutions:

- Take pseudo-inverse. \rightarrow general.

- add regularization.

- manually clean up correlated features. \rightarrow for ①.

- remove some features/add more examples \rightarrow for ②.