§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

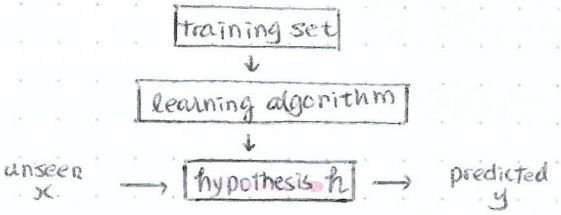
(parametric)

SUPERVISED LEARNING / REGRESSION

⟨Ex⟩ Housing price prediction:

	x_1 size (ft^2)	x_2 #bedrooms	x_j ...	y price ($1000s)
$(x^{(1)}, y^{(1)})$	2104	3		400
$(x^{(2)}, y^{(2)})$	1600	3	...	330
$(x^{(3)}, y^{(3)})$	2400	3		369
	⋮	⋮		⋮

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

$x_j^{(i)}$: i → index into training examples, $i=1,\dots,m$; j → index into features, $j=(0),1,\dots,n$.

(eg) $x_1^{(1)} = 2104$, $x_1^{(2)} = 1600$.

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 (linear function + intercept / translation).

↳ shorthand: $h(x)$.

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

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

Notes.

- linear regression ⇒ linear in parameters.
  (eg) $h(x) = \theta_0 + \theta_1 x + \theta_2 x^2$ ⇒ linear model (V).
  (x_1, x_2)
- (Univariate) linear regression: $y \in$ scalar / Simple linear regression: $x \in$ scalar.
  \ Multiple linear regression: $x \in$ vector.
  Multivariate linear regression: $y \in$ vector.

SQUARED ERROR COST FUNCTION

How to choose "good" θ values? ⇒ As a first attempt: choose θ s.t. $h(x) \approx y$ for training examples. (best E_{in} ≠ best E_{out})

Define quantitative measure of model performance:

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

⇒ Choose ℓ_2 loss: $\boxed{\ell(h_\theta(x), y) = \frac{1}{2}(h_\theta(x) - y)^2}$.

↳ convention (∵ derivatives of ℓ_2).

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

⇒ Squared error cost function:

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

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

Notes:

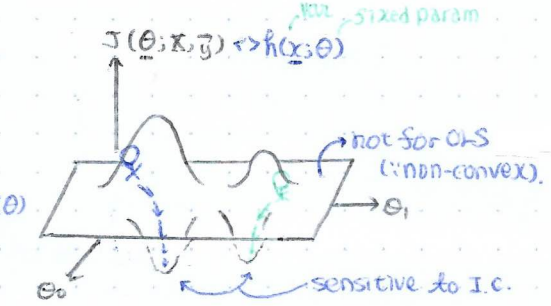
- linear regression + MSE = ordinary least squares (OLS).
- Why ℓ_2? ⇒ There are other forms of loss (eg. ℓ_1 loss $|h_\theta(x) - y|$), but ℓ_2 is a common choice for regression problems:
  - ↳ works well.
  - ↳ math convenience (continuously differentiable).
  - ↳ follows naturally from $y^{(i)} \sim$ 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.

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

$:=$ assignment ($a := a+1, \forall$) ↔ truth assertion ($a = a+1, \times$).

α learning rate $\in \mathbb{R}^+$

For OLS:

Consider 1 example (∵ linearity of differentiation):

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

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

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

⇒ Least mean squares (LMS) update rule: ← OLS + GD.

repeat until convergence {

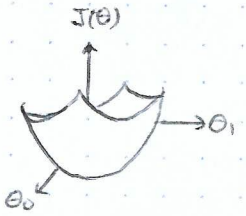
$$\theta_j := \theta_j - \alpha \sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}, \forall j.$$

}

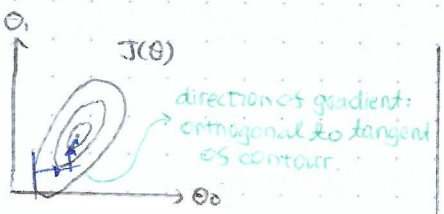
$$\theta := \theta - \frac{\alpha}{m} X^T (\vec{h} - \vec{y}).$$

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

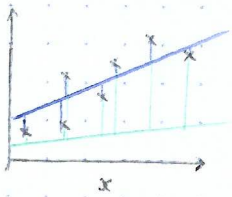
For OLS, $J(\theta)$ is a convex / quadratic fx $\Rightarrow$ 1 global min.



3D surface: bowl-shaped



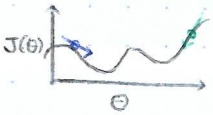
Contours: ellipses.



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

Properties

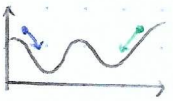
- Minus sign:



Slope $< 0 \Rightarrow$ move right.
Slope $> 0 \Rightarrow$ move left.

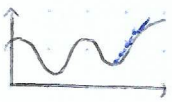
For maximization $\Rightarrow$ Gradient ascent:
$\theta := \theta + \alpha \vec{\nabla} J(\theta)$.

- Local search:



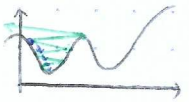
Depending on initialization, may end up at different local minimum.
(or wherever $\vec{\nabla} J(\theta) = \vec{0}$)

- Adaptive stepsize:



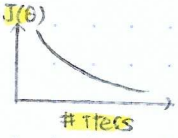
When approaching min, $|\vec{\nabla} J(\theta)| \downarrow$, automatically take smaller steps.
$\Rightarrow$ GD can converge w. a fixed α.

- 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 $\theta^{(t+1)}$ differs significantly from $\theta^{(t)}$
(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.
- $\because$ 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

$\sum_{i=1}^{m}$

The algorithm introduced above is called "batch gradient descent (BGD)".
$\because$ It scans through the entire training set to take a single step. $\Rightarrow$ slow for big data.

Alternative: stochastic / incremental gradient descent (SGD).

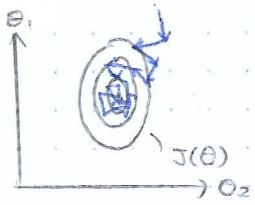
Repeat until convergence {
  For $i = 1, \dots, m$ {
    $\theta_j := \theta_j - \alpha \frac{\partial J^{(i)}(\theta)}{\partial \theta_j}$, $\forall j$.
  }
}

cost / loss for 1 example:
(eg) $J^{(i)}(\theta) = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2$

(+) 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.
↳ $>10^2, 10^3$

(-): 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. ↳ tho 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
still be

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).

for $A \in \mathbb{R}^{m \times n}$, $f(A): \mathbb{R}^{m \times n} \mapsto \mathbb{R}$,

$$\nabla_A f(A) = \begin{bmatrix} \frac{\partial f}{\partial A_{11}} & \cdots & \frac{\partial f}{\partial A_{1n}} \\ \vdots & & \\ \frac{\partial f}{\partial A_{m1}} & \cdots & \frac{\partial f}{\partial A_{mn}} \end{bmatrix} \in \mathbb{R}^{m \times n}$$

↳ 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), $\mathrm{Tr}(A) = \sum_{i=1}^{n} A_{ii}$.

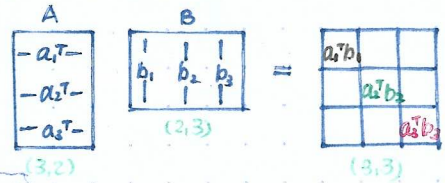
① $\mathrm{Tr}\, A = \mathrm{Tr}\, A^T$. → ∵ flipping A doesn't change the diagonal elements.

② $\nabla_A \mathrm{Tr}\, AB = B^T$. → (cf) $\frac{d(ab)}{da} = b$ (+ match dim),
$(m,n)(n,m) \to (m,m)$, ✓
$f(A)$; B: fixed param.

③ Invariant under cyclic permutations →

$$\mathrm{Tr}\, AB = \mathrm{Tr}\, BA$$
$(m,n)(n,m) \to (m,m)$ $(n,m)(m,n) \to (n,n)$

$$\mathrm{Tr}\, ABC = \mathrm{Tr}\, BCA = \mathrm{Tr}\, CAB$$



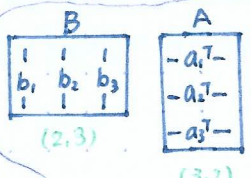
distribute element multiplication in different ways ⇒ same Tr.

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

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

④ $\nabla_A \mathrm{Tr}\, AA^TC = CA + C^TA$.
$f(A)$ $(m,m)(m,n)$ $(n,m)(m,n) \to (m,n)$ ✓
$(m,n)(n,m)(m,m)$

→ (cf) $\frac{d(a^2c)}{da} = 2ac$.



$= b_1a_1^T + b_2a_2^T + b_3a_3^T =$ (2,2)

Proof

- step 1: express $J(\Theta)$ in array notation.

Define:

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

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

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 a: Compute $\nabla_\theta J(\theta)$ using Tr properties & set to zero.

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

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

$\to 0$ (∵ independent of θ).

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

$\to s = \mathrm{Tr}\,s$.

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

For (a):

$\nabla_\theta \mathrm{Tr}\,\theta^T X^T X\theta$ ③

$= \nabla_\theta \mathrm{Tr}\,\theta\theta^T X^T X$ (A, A^T, C) ④

$= X^T X\theta + X^T X\theta$

$= 2X^T X\theta$.

For (b):

$\nabla_\theta \mathrm{Tr}\,\vec{y}^T X\theta$ ③

$= \nabla_\theta \mathrm{Tr}\,\theta\,\vec{y}^T X$ (A, B) ②

$= X^T\vec{y}$.

For (c):

$\nabla_\theta \mathrm{Tr}\,\theta^T X^T \vec{y}$ $(\mathrm{Tr})S = (\mathrm{Tr})S^T$ OR ①

$= \nabla_\theta \mathrm{Tr}\,\vec{y}^T X\theta$

$= $ (b).

$$\Rightarrow \nabla_\theta J(\theta) = X^T X\theta - X^T\vec{y} \triangleq \vec{0}$$ "Normal equation"

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

$(n+1, n+1)\ (n+1, m)\ (m, 1) \to (n+1, 1)$, ✓

Notes

- ∵ Normal eqn involves solving $(X^T X)^{-1} \sim \mathcal{O}(n^3)$
  ⇒ For large n, GD is preferred.
- When $(X^T X)$ is non-invertible ⇒ redundant features (linearly dependent).
  ↳ OR: n is too large for given m.

Solutions:

↳ manually clean up correlated features.

↳ take pseudo-inverse.

OR: use regularization.