Sep 18th, 2024

Stochastic Gradient Descent (SGD)

example: $f(w) = w^2 - 6w + 11$ ← minimize

$= (w-3)^2 + 2$

↓

have equal zero, so minimum is $w = 3$

$\Rightarrow f'(w) = 2w - 6$

SGD: Taking deriv one value at a time

How to update: $w \leftarrow w - \alpha f'(w)$

↑ travel in opposite direction of deriv

α ← the step size (how much of an update)

What to do?

ex.

step ① $w = 0$; $\alpha = 0.1$ $f'(w) = 2w - 6$

$w \leftarrow 0 - 0.1(2w - 6) = 0 - 0.1(0 - 6) = 0.6$

> w is updated to 0.6 (added 0.6)

step ② $w = 0.6$; $\alpha = 0.1$

$w \leftarrow 0.6 - 0.1(2(0.6) - 6) = 0.6 - 0.1(1.2 - 6)$

$= 0.6 - 0.1(-4.8) = 0.6 + 0.48 = \boxed{1.08}$

> w is updated

[Will hopefully get to value to minimize cost function]

When do we stop?

→ based on goal / we decide

$w_0 = 0$, $w_1 = 0.6$, $w_2 = 1.08$

evaluate $f(w) \rightarrow f(w_0)$ etc + look at differences/change

ε ← lowest change we should go to ε

$|f(w^t) - f(w^{t-1})| < \varepsilon$, $\varepsilon = 1 \times 10^{-8}$ for example

SGD for Linear Regression

→ function is different (we use the cost function)

→ key idea is same: taking deriv one data point at a time

$$\nabla J_{\vec{x}_i} = \frac{d\, J(\vec{w})_{\vec{x}_i}}{d\vec{w}} = (\vec{w} \cdot \vec{x}_i - y_i)\, \vec{x}_i$$

↑ one data point ($\vec{x}_i$), not whole sum

→ approx.

What to do?

→ make pre-determined max steps ← fail safe

→ iterate through data points

→ for each data point, update w w/ deriv

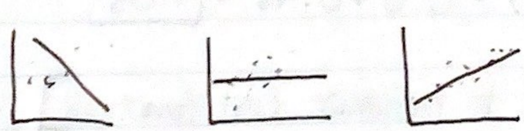
$$\vec{w} \leftarrow \vec{w} - \alpha (\vec{w} \cdot \vec{x}_i - y_i)\, \vec{x}_i$$

→ check for convergence:

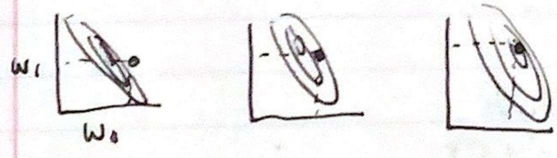
$$\left| J(\vec{w}^{\,t}) - J(\vec{w}^{\,t-1}) \right| < \varepsilon$$

real way you should be breaking out of loop

linear Model + Cost function J



← process makes model better



changes to weight values

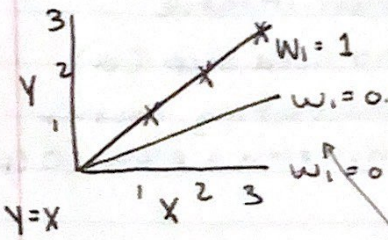
Cost Function (extra Practice)

form of model → $h_w(x) = W_1 x$ (assume $w_0 = 0$)

cost function: $J = \frac{1}{2}\sum_{i=1}^{3}(y_i - w_1 x_i)^2$

weight values:

$W_1 = 0$ $W_1 = 0.5$



x = data point

Calculating cost function value:

$J(0) = \frac{1}{2}\sum (y_i - 0x_i)^2$

$= \frac{1}{2}\left((1-0(1))^2 + (2-0(2))^2 + (3-0(3))^2\right)$

$= \frac{1}{2}(1+4+9)$

$= \boxed{7}$

$J(0.5) = \frac{1}{2}\sum (y_i - 0.5x_i)^2$

$= \frac{1}{2}\left((1-0.5)^2 + (2-1)^2 + (3-1.5)^2\right)$

$= \boxed{1.75}$

Choosing step size (α)

→ ~~[illegible]~~ too small: slow convergence

→ too large: increasing value for $J(w)$; may overshoot minimum; may fail to converge

---

Handout data:

it 1: $w_1 = 0$, $J(\theta) = 0.41$

it 2: ~~[illegible]~~, $J(\theta) = 0.35$

it 12: $J(\theta)$ 0.15

it 100: $J(\theta)$ approaching 0

↓ slope gets better + cost converges

Handout 6 notes:

$x_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, $x_2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$, model $h = W_0 + W_1 X$, $\vec{w} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}$

$\vec{w} \cdot \vec{x_i} = \vec{\hat{y}_i}$ ← prediction

$X = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$ add 1s to allow for vector * and +

vector vector

→ plug points into equation: $\vec{w} \leftarrow \vec{w} - \alpha(\vec{w} \cdot x_i - y_i) x_i$

Pros + Cons

GD:	Normal:
- requires iteration	- non-iterative
- need step (α)	- no need step (α)
- when p is large, it works better	- calculating matrix inversion is slow ($O(p^3)$)

- can support online learning
  > we can easily update data + weights