=) Mini-batch gradient descent

Vectorization allows you to efficiently compute on m examples.

$X = [x^{(1)}, x^{(2)}, \dots, x^{(m)}]$ (n_x, m)

$X^{\{1\}}, X^{\{2\}} \dots X^{\{5000\}}$

$Y = [y^{(1)}, y^{(2)}, y^{(3)} \dots y^{(m)}]$ $(1, m)$

$Y^{\{1\}}, Y^{\{2\}} \dots Y^{\{5000\}}$

What if $m = 5.000.000$?

mini-batches of 1.000 each

$x^{(i)} \rightarrow$ i trainig example

$z^{[l]} \rightarrow$ layer

$X^{\{t\}}, Y\{t\} \rightarrow$ mini-batch

Batch => entire training set same time

mini-batch => single mini batch, $X^{\{t\}}, Y\{t\}$

Mini-batch gradien descent

for $t = 1, \dots 5000$

Forward prop on $X^{\{t\}}$

$$Z^{[1]} = W^{[1]} X^{\{t\}} + b^{[1]}$$
$$A^{[1]} = g^{[1]}(Z^{[1]})$$
$$\vdots$$
$$A^{[L]} = g^{[L]}(Z^{[L]})$$

} Vectorized implementation → from $X^{\{t\}}, Y^{\{t\}}$

Compute cost $J = \frac{1}{1000} \sum_{i=1}^{l} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \underbrace{\frac{\lambda}{2.1000} \sum_{e} \|w^{[e]}\|_F^2}_{\text{regularization}}$

Backprop to compute gradients (dw, db) $J^{\{t\}}$ using $(X^{\{t\}}, Y^{\{t\}})$

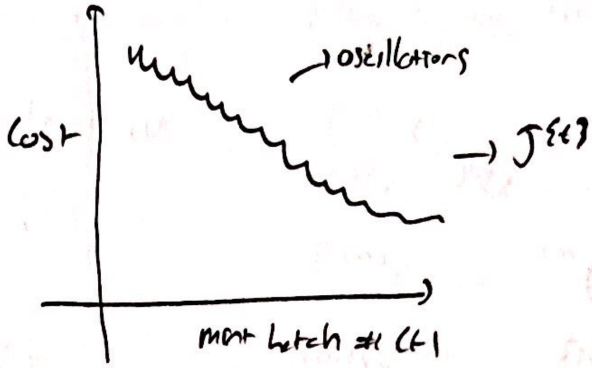
$W^{[e]} := W^{[e]} - \alpha\, dw^{[e]}$, $b^{[e]} := b^{[e]} - \alpha\, db^{[e]}$

"1 epoch" → pass through training set

Understanding mini-batch gradient descent

Batch gradient descent

Mini-batch gradient descent

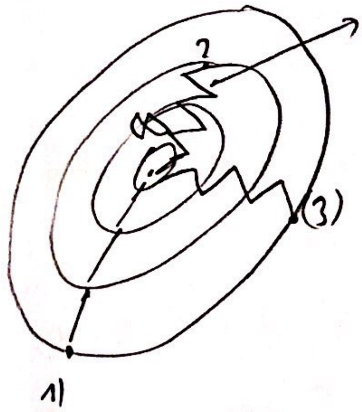


* Choosing your mini-batch size #

1) If mini-batch size = m : Batch gradient desent. $(X^{\{1\}}, Y^{\{1\}}) = (X, Y)$

2) If mini-batch size = 1 : Stochastic gradient descent: Every example is its own mini-batch

$(X^{\{1\}}, Y^{\{1\}}) = (x^{(1)}, y^{(1)})$



In practice: Somewhere in-between 1 and m

Batch gradient desent	Stochastic gradient desent
↓ Too long per iteration	↓ lose speed up from vectori:

↓ In-between (mini-batch size) not big not small

↓ Fastest learning

- vectorization
- Make progress without entire traing set

Choosing your mini-batch size

- If small training set: use batch gradient des. ($m \leq 2000$)
- If bigger training set: Typical mini-batch size. → (64, 128, 256, 512) 2^n ← computer. Common 1024

Make sure minibatch fits in CPU/GPU memory $X^{\{t\}}, Y^{\{t\}}$

Exponentially weighted averages

$V_t = \beta V_{t-1} + (1-\beta)\theta_t$

$\beta = 0.9$: ≈ 10 days ten — $\frac{1}{1-\beta}$ days temperature

$\beta = 0.98$: ≈ 50 days ten

$\beta = 0.5$: ≈ 2 days ten — $\frac{1}{1-0.98} = 50$ days.

$V_t = \beta V_{t-1} + (1-\beta)\theta_t$

$V_{100} = 0.1\theta_{100} + 0.9(V_{99} = 0.1\theta_{99} + 0.9(V_{98} = 0.1\theta_{98} + 0.9 V_{97}$

$= 0.1\theta_{100} + 0.1 \times 0.9\,\theta_{99} + 0.1(0.9)^2\theta_{98} + 0.1(0.9)^2\theta_{97} \ldots$

$0.9^{10} \approx 0.35 \approx \frac{1}{e}$ — $\varepsilon = 0.1$ in this example — $(1-\varepsilon)^{1/\varepsilon} = \frac{1}{e}$ — $\varepsilon = 1-\beta$ — $0.98^{50} \approx \frac{1}{e}$

Implementing exponentially weighted averages

$V_\theta := 0$

$V_\theta := \beta V_\theta + (1-\beta)\theta_1$

$V_\theta := \beta V_\theta + (1-\beta)\theta_2$

$V_\theta = 0$

Repeat {

Get next θ_t

$V_\theta := \beta V_\theta + (1-\beta)\theta_t$ }

Bölüm

, correction in exponentially weighted average

Bias correction ile $v_0=0$ sorunu çözül

$v_0=0$ $\beta=0.98$

$v_1=0.98 \cancel{v_0}+0.02\theta_1$

$v_2=0.98v_1+0.02\theta_2$

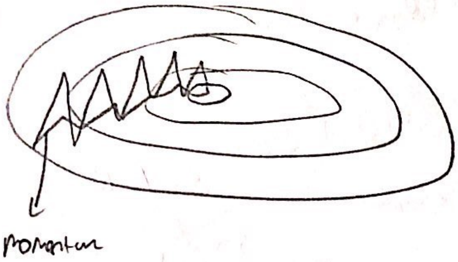
v_0' 0 olduğu için ilk kısımlar hatalı çıkacak!!

$$\frac{v_t}{1-\beta^t}$$

$t=2:$ $1-\beta^t=1-(0.98)^2=0.0396$

$$\frac{v_2}{0.0396}=\frac{0.0196\theta_1+0.02\theta_2}{0.0396}$$

Gradient descent with momentum



Momentum

On iteration t:

compute dw, db on current mini-batch

$V_{dw}=\beta V_{dw}+\cancel{(1-\beta)}dw$? often omitted

$V_{db}=\beta V_{db}+\cancel{(1-\beta)}db$ Hoa kullanıyo

$w=w-\alpha V_{dw}$, $b:=b-\alpha V_{db}$ $(1-\beta)$

"$v_\theta=\beta v_\theta+(1-\beta)\theta_t$

$\frac{V_{dw}}{1-\beta^t}$ => bias correction kullanılmaz

Hyperparameters: α, β $\beta=0.9$

learning rate

RMSprop (Root Mean Squared)

Bunun için RMSprop kullanılır.

On iteration t:

Compute dw, db on current mini-batch

$S_{dw} = \beta S_{dw} + (1-\beta) dw^2$ → element wise

$S_{db} = \beta S_{db} + (1-\beta) db^2$

$w := w - \alpha \frac{dw}{\sqrt{S_{dw}} + \varepsilon}$ → küçük olmalı

$b := b - \alpha \frac{db}{\sqrt{S_{db}}}$ → büyük olmalı

$\varepsilon \rightarrow 10^{-7}$ sıfır olmasın diye

Adam optimization algorithm

(momentum + RMSprop)

$V_{dw} = 0$, $S_{dw} = 0$ $V_{db}, S_{db} = 0$

On iteration t:

Compute dw, db using current mini batch

$V_{dw} = \beta_1 V_{dw} + (1-\beta_1) dw$, $V_{db} = \beta_1 V_{db} + (1-\beta_1) db$ → momentum

$S_{dw} = \beta_2 V_{dw} + (1-\beta_2) dw^2$ $S_{db} = \beta_2 V_{db} + (1-\beta_2) db^2$ →

bias correction

$V_{dw}^{corrected} = V_{dw} / (1-\beta_1^t)$, $V_{db}^{corrected} = V_{db} / (1-\beta_1^t)$

$S_{dw}^{corrected} = S_{dw} / (1-\beta_2^t)$, $S_{db}^{corrected} = S_{db} / (1-\beta_2^t)$

$w := w - \alpha \frac{V_{dw}^{corr.}}{\sqrt{S_{dw}^{corr.}} + \varepsilon}$ $b := b - \alpha \frac{V_{db}^{corrected}}{\sqrt{S_{db}^{corr}} + \varepsilon}$

$\varepsilon = 10^{-8}$

- Hyperparameters choice

Adam : Adaptive Moment Estimation

α needs to be tune

$\beta_1 : 0.9$ (dw) → First moment

$\beta_2 : 0.999$ (dw^2) → Second moment

$\varepsilon : 10^{-8}$ (not important)

Learning rate decay

Hedefe yaklaştıkça küçültülmeli gerek

1 epoch = 1 pass through data

$$\alpha = \frac{1}{1 + \text{decay-rate} * \text{epoch-num}} \alpha_0$$

Epoch	α
1	0.1
2	0.67
3	0.5
4	0.4

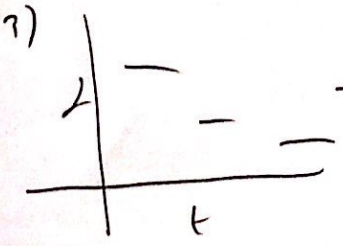
$\alpha_0 = 0.2$
decay rate = 1

Hedefe yaklaştıkça büyük learning rate gitmeden küçülebilir minimum'u

Other learning rate decay methods #

1) $\alpha = 0.95^{\text{epoch-num}} \cdot \alpha_0$ → exponentially decay Manel decay ele indiriyor.

2) $\alpha = \frac{k}{\sqrt{\text{epochnum}}} \cdot \alpha_0$ or $\frac{k}{\sqrt{t}} \cdot \alpha_0$

3) 

→ discrete staircase