Mini-Batch Gradient Descent

e.g. $m = 1{,}000{,}000$

→ split into 1000 mini-batches of 1,000 each

Mini-Batch t : $X^{\{t\}}$, $Y^{\{t\}}$
$(n_x, 1000)$ $(1, 1000)$

for $t = 1$ to 1000:

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

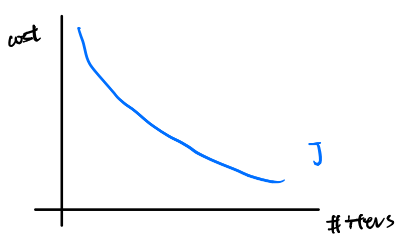
$$\left.\begin{aligned} Z^{[1]} &= W^{[1]} X^{\{t\}} + b^{[1]} \\ A^{[1]} &= g^{[1]}(Z^{[1]}) \\ &\vdots \\ A^{[L]} &= g^{[L]}(Z^{[L]}) \end{aligned}\right\} \text{Vectorized implementation}$$

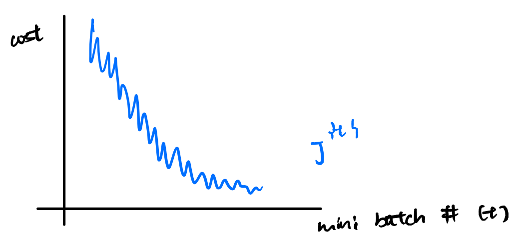
Compute $J^{\{t\}} = \frac{1}{1000} \sum_{i=1}^{1000} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2 \cdot 1000} \sum_{l} \| w^{[l]} \|_F^2$

Backprop to compute gradient w.r.t. $J^{\{t\}}$ (using $(X^{\{t\}}, Y^{\{t\}})$)

$W^{[l]} := W^{[l]} - \alpha dW^{[l]}$; $b^{[l]} := b^{[l]} - \alpha b^{[l]}$

"1 epoch" ~ pass through training set





Batch Gradient Descent $\xleftarrow{\text{size} = m}$ Mini Gradient Descent

Stochastic ~ ↙ size = 1

(lose speedup from vectorization)

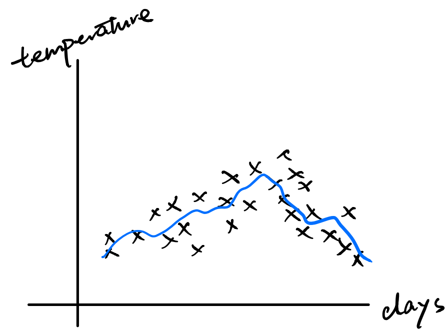
⇒ typical mini-batch size :

64 , 128 , 256 , . . .

Make sure mini-batch fit in CPU / GPU memory

Exponentially weighted Averages

e.g. $\theta_1 = 40°F$
$\theta_2 = 49°F$
$\vdots$

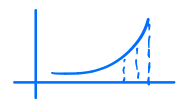


$\Rightarrow$ $V_0 = 0$
$V_1 = 0.9 V_0 + 0.1\ \theta_1$
$V_2 = 0.9\ V_1 + 0.1\ \theta_2$
$\vdots$
$V_t = 0.9\ V_{t-1} + 0.1\ \theta_t$

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

averages over $\approx \frac{1}{1-\beta}$ days

e.g. $V_{100} = 0.1\ \theta_{100} + 0.1 \times 0.9\ \theta_{99} + 0.1 \times (0.9)^2\ \theta_{98} + \cdots$



$0.9^{10} \approx 0.35 \approx \frac{1}{e}$

$(1-\varepsilon)^{1/\varepsilon} = \frac{1}{e}$

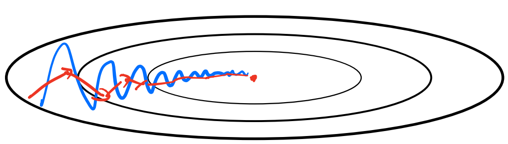
after $\frac{1}{1-\beta}$ days. about 1/3 left

$\Rightarrow$ $V_\theta = 0$
Repeat {
 Get next θ_t
 $V_\theta := \beta V_\theta + (1-\beta)\theta_t$
}

Problem:
$V_1 = \beta V_0 + (1-\beta)\ \theta_1$
$= (1-\beta)\ \theta_1$

Bias Correction $\Rightarrow \frac{V_t}{1-\beta^t}$
(only matters in early stage)

Gradient Descent w/ Momentum



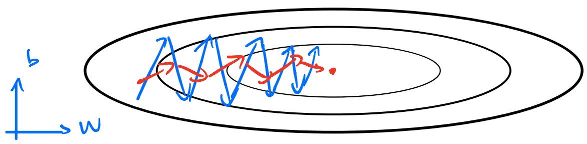
$\updownarrow$ slower
$\leftrightarrow$ faster

Momentum:
On iteration t:
 Compute dW, db on current mini-batch
 $V_{dW} = \beta\ V_{dW} + (1-\beta)\ dW$
 $V_{db} = \beta\ V_{db} + (1-\beta)\ db$
 $W := W - \alpha\ V_{dW}$, $b := b - \alpha\ V_{db}$

another form:
$\beta V_{dW} + dW$ ← tune α

(α, β hyperparameters)

RMSProp



↕ slower

↔ faster

On iteration t,

compute dw, db on current mini-batch

$$S_{dw} = \beta_2 S_{dw} + (1-\beta_2)\, dw^2 \quad \leftarrow \text{small}$$

$$S_{db} = \beta_2 S_{db} + (1-\beta_2)\, db^2 \quad \leftarrow \text{large}$$

$$w := w - \alpha \frac{dw}{\sqrt{S_{dw}+\varepsilon}} \qquad b := b - \alpha \frac{db}{\sqrt{S_{db}+\varepsilon}}$$

Prevent zero error

e.g. $\varepsilon = 10^{-8}$

Adam Optimization Algorithm — Adaptive Moment Estimation

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

On iteration t,

Compute dw, db using current mini-batch

$$V_{dw} = \beta_1 V_{dw} + (1-\beta_1)\, dw, \quad V_{db} = \beta_1 V_{db} + (1-\beta_1)\, db \quad \leftarrow \text{Momentum}$$

$$S_{dw} = \beta_2 S_{dw} + (1-\beta_2)\, dw^2, \quad S_{db} = \beta_2 S_{db} + (1-\beta_2)\, db^2 \quad \leftarrow \text{RMSProp}$$

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

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

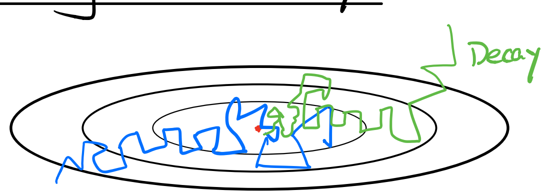
$$\Rightarrow \quad w := w - \alpha \frac{V_{dw}^{corrected}}{\sqrt{S_{dw}^{corrected}} + \varepsilon}$$

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

hyperparameters:

- α: needs to be tuned
- β_1: 0.9 (dw)
- β_2: 0.999 (dw^2)
- ε: 10^{-8}

Learning Rate Decay



1 epoch = 1 pass of data

$$\alpha = \frac{1}{1 + \underbrace{\text{decay rate}}_{1} * \text{epoch \#}} \overset{0.2}{\alpha_0}$$

$\Rightarrow$ e.g.

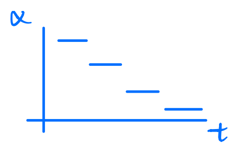
epoch	1	2	3	4	...
α	0.1	0.067	0.05	0.04	...

Other decay methods:

$$\alpha = 0.95^{\text{epoch \#}} \alpha_0$$

$$\alpha = \frac{k}{\sqrt{\text{epoch \#}}} \alpha_0$$

$\alpha =$



Problem of Local Optima

In high dimension, saddle points are more common

Unlikely to stuck in a bad local optima

Plateaus can make learning slow