Batch vs Mini-Batch gradient descent

- Vectorization allows us to compute on m examples effectively
- If we have a really large number of trainy examples, we split them into small parts mini-batches

$X = \left[x^{(1)} \ldots x^{(1000)} \mid x^{(1001)} \ldots x^{(2000)} \mid \ldots \mid \ldots x^{(m)} \right]$

$X^{\{1\}}$ $(n_x, 1000)$ → mini-batch

$X^{\{2\}}$

$X^{\{5000\}}$ → same for y

Mini-batch gradient descent

repeat {

for $t = 1, \ldots, 5000$ — 1 step of gradient descent for $X^{\{t\}}, y^{\{t\}}$

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

$Z^{[1]} = W^{[1]} X^{\{t\}} + b^{[1]}$

$A^{[1]} = g^{[1]}(Z^{[1]})$

$\vdots$

$A^{[L]} = g^{[L]}(Z^{[L]})$

compute cost $J = \frac{1}{1000} \sum_{i=1}^{l} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \text{regularization}$

back prop to compute gradients

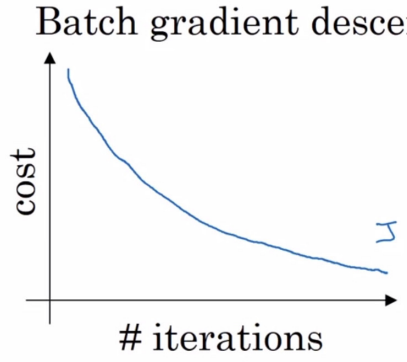
$W^{[L]} := W^{[L]} - \alpha \, dW^{[L]}$, $b^{[L]} := b^{[L]} - \alpha \, db^{[L]}$

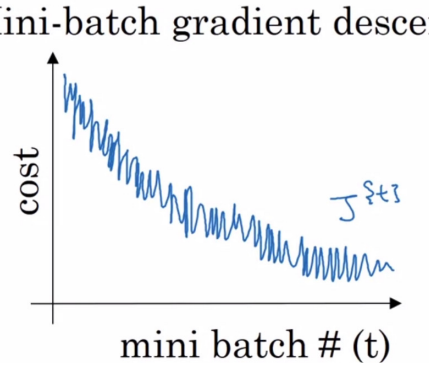
}

}

→ "1 epoch"

↳ single pass through training set





- Minibatch where every training example is its own mini-batch is called Stochastic Gradient Descent it never hits minimum.

Stochastic	In-between	Batch
→ Lose speed from losing vectorization	→ Fastest learning → Vectorization → Make progress without processing entire training set	→ Too long per iteration → vectorized → all at once

Choosing Mini-batch size

If small training set: use batch gradient descent if $m \leq 2000$

Typical mini-batch sizes: 64, 128, 256, 512 (power of 2)

* make sure mini-batch fits in cpu/gpu memory → mini-batch size is a hyperparameter

Exponentially Weighted Averages

$$v_t = \beta v_{t-1} + (1-\beta)\theta_t$$

Implementing:

$v_0 = 0$

repeat {

get next θ_t

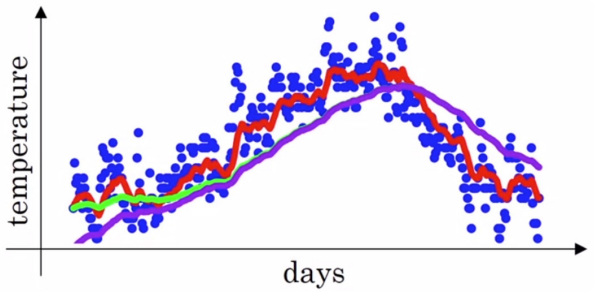
$v_\theta := \beta v_\theta + (1-\beta)\theta_t$

}

Bias Correction

- The curve starts of really low and that is a problem
- We use $\beta = \frac{v_t}{1-\beta^t}$ because it helps get rid of the bias as over examples β approaches zero and the purple line becomes the same as green line

(before fixing bias) (after fixing bias)



Gradient Descent with Momentum

Momentum: leads to much smaller steps and gradient descent is faster

On iteration t:

$V_{dW}, V_{db} = 0$

Compute dW, db on current mini-batch / batch

$V_{dW} = \beta V_{dW} + (1-\beta) V_{dW}$

$V_{db} = \beta V_{db} + (1-\beta) V_{db}$ — β: friction; $(1-\beta)$: acceleration; V: velocity

$W = W - \alpha V_{dW}$

$b = b - \alpha V_{db}$

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

the β helps the gradient descent move roughly in the speed it was moving in already

0.9 is the most common value for β

Bias correction → $\frac{V_{dW}}{1-\beta_t}$

RMSprop

On iteration t:

Compute dW, db on current mini-batch

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

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

$W = W - \alpha \frac{dW}{\sqrt{S_{dW}} + \varepsilon}$

$b = b - \alpha \frac{db}{\sqrt{S_{db}} + \varepsilon}$

$\varepsilon = 10^{-8}$ for numerical stability

Adam Optimization Algorithm

$V_{dW} = 0, \; S_{dW} = 0, \; V_{db} = 0, \; S_{db} = 0$

On iteration t:

Compute dW, db, using current mini-batch

$V_{dW} = \beta_1 V_{dW} + (1-\beta_1) dW$

$V_{db} = \beta_1 V_{db} + (1-\beta_1) db$

"momentum" β_1

$S_{dW} = \beta_2 S_{dW} + (1-\beta_2) dW^2$

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

"RMSprop" β_2

$$V_{dw}^{corrected} = \frac{V_{dw}}{(1-\beta_1^t)}$$

$$V_{db}^{corrected} = \frac{V_{db}}{(1-\beta_2^t)}$$

$$S_{dw}^{corrected} = \frac{S_{dw}}{(1-\beta_1^t)}$$

$$S_{db}^{corrected} = \frac{S_{db}}{(1-\beta_2^t)}$$

"bias correction"

$$W = W - \alpha \frac{V_{dw}^{corrected}}{\sqrt{S_{dw}^{corrected}} + \varepsilon}$$

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

update

Hyperparameter Choice

α : needs to be tuned

β_1 : 0.9 (momentum)

β_2 : 0.999 (RMS prop)

ε : 10^{-8} recommended

ADAM: Adaptive Moment Estimation

Learning Rate Decay

1 epoch = 1 pass through data

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

decay-rate → hyperparameter

- Learning rate gradually decreases which helps us take smaller and smaller steps to converge to the minima.

Other ways -

→ $\alpha = 0.95^{\text{epochnum}} \cdot \alpha_0$ – Exponential Decay

0.95 → some number less than 1

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

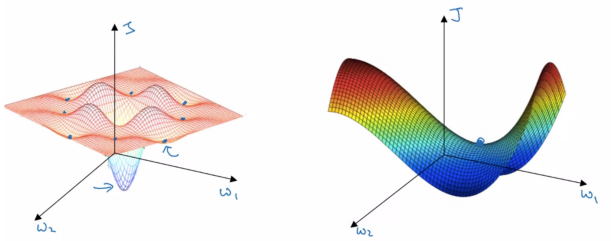
k → constant

→ decreasing learning rate by a factor "discrete staircase"

→ Manual decay

Local Optima Problem

- Most local optima are saddle points
- The chances of all n parameters to be 0 is really small and there is often a curve



Problem of Plateau

- unlikely to get stuck in a bad local optima
- plateaus make learning slow (methods like ADAM help)