§ Improving DNN — Week 2 Optimization Algorithms

1. Batch VS. mini-batch

$X = [\underbrace{x^{(1)}\ x^{(2)}\ x^{(3)}\ \cdots\ x^{(1000)}}_{X^{\{1\}}}\ \underbrace{x^{(1001)}\ \cdots\ x^{(2000)}}_{X^{\{2\}}}\ \cdots\ \underbrace{x^{(m)}}_{X^{\{5000\}}}]$ (n_x, m)

$Y = [\underbrace{y^{(1)}\ y^{(2)}\ y^{(3)}\ \cdots\ y^{(1000)}}_{Y^{\{1\}}}\ \underbrace{y^{(1001)}\ \cdots\ y^{(2000)}}_{Y^{\{2\}}}\ \cdots\ \underbrace{y^{(m)}}_{Y^{\{5000\}}}]$ $(1, m)$

$5000, 000$

for $t = 1, \cdots, 5000$

- forward prop on $X^{\{t\}}$ (1000 examples)

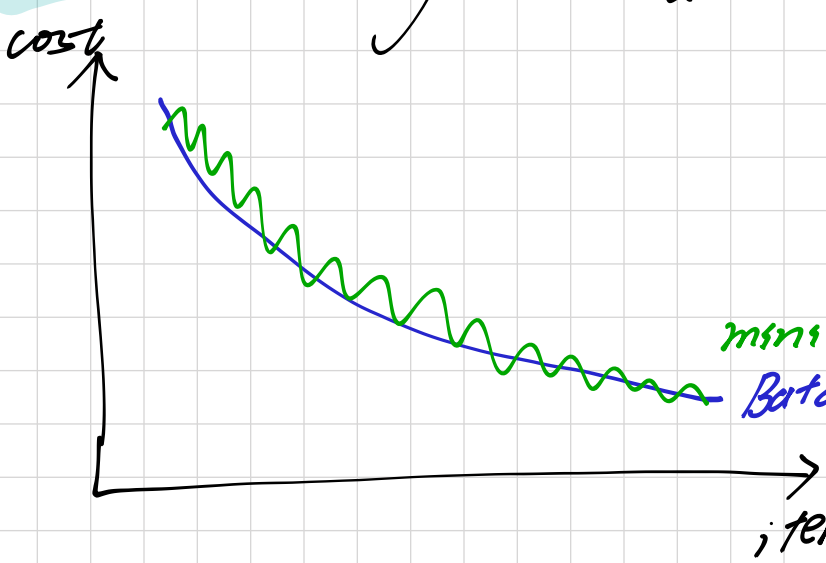
$$Z^{[1]} = W^{[1]} X^{\{t\}} + b^{[1]}$$
$$A^{[1]} = g^{[1]}(Z^{[1]})$$
$$\vdots$$
$$A^{[L]} = g^{[L]}(Z^{[L]})$$

- compute cost $J = \frac{1}{1000} \sum_{i=1}^{l} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2 \cdot 1000} \sum_l \|W^{[l]}\|_F^2$ from $X^{\{t\}}, Y^{\{t\}}$
- Backprop to compute cost $J^{\{t\}}$ (using $(X^{\{t\}}, Y^{\{t\}})$)
- $W^{[l]} := W^{[l]} - \alpha dW^{[l]}$
- $b^{[l]} := b^{[l]} - \alpha db^{[l]}$

1 step of gradient descent using $X^{\{t\}}, Y^{\{t\}}$ (as if $m = 1000$)

→ 1 epoch → passing through training set

2. mini-batch gradient descent



plot $J^{\{t\}}$ using $X^{\{t\}}, Y^{\{t\}}$

If mini-batch size = m : Batch gradient descent

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

If mini-batch size = 1 : Stochastic gradient descent

$(X^{\{1\}}, Y^{\{1\}}) = (X^{(1)}, y^{(1)}) \cdots (X^{(4)}, Y^{(4)})$

In practice : between 1 and m

Batch gradient descent → too long per iteration

In between → faster learning

Stochastic gradient descent → lose vectorization

{small training set : use batch gradient descent ($m \leq 2000$)

Typical mini-batch size : 64, 128, 256, 512, 1024

☆ Make sure mini-batch fit in CPU/GPU memory

3. Exponentially weighted averages

$v_0 = 0$

$v_1 = \beta v_0 + (1-\beta)\theta_1$

$v_2 = \beta v_1 + (1-\beta)\theta_2$

$v_3 = \beta v_2 + (1-\beta)\theta_3$

...

$V_\theta := 0$

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

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

⋮

$V_\theta = 0$

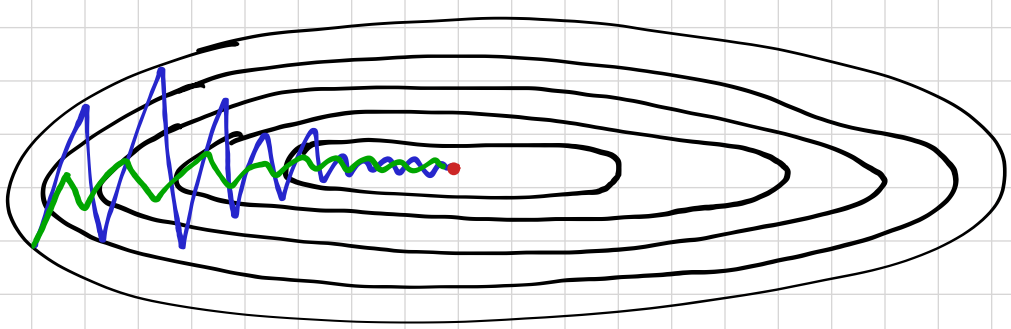
Repeat {

Get next θ_t

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

}

4. Gradient Descent with momentum



if wants ↕ slower learning

⟷ faster learning

⇒ use momentum

Momentum:

On iteration t:

Compute dw, db on current mini-batch

$$V_{dw} = \beta V_{dw} + (1-\beta) dw$$
$$V_{db} = \beta V_{db} + (1-\beta) db$$
$$\Leftarrow V_\theta = \beta V_{\theta-1} + (1-\beta)\theta_t$$

$$w := w - \alpha V_{dw}$$
$$b = b - \alpha V_{db}$$

friction — velocity — acceration

5. RMSprop

On iteration t:

Compute dw, db on mini-batch

$$S_{dw} = \beta S_{dw} + (1-\beta) dw^2$$
$$S_{db} = \beta S_{db} + (1-\beta) db^2$$
$$w := w - \alpha \frac{dw}{\sqrt{S_{dw}} + \varepsilon} \quad \varepsilon = 10^{-8}$$
$$b := b - \alpha \frac{db}{\sqrt{S_{db}} + \varepsilon}$$

6. 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

$$V_{dw} = \beta_1 V_{dw} + (1-\beta_1) dw$$
$$V_{db} = \beta_1 V_{db} + (1-\beta_1) db$$
$$S_{dw} = \beta_2 S_{dw} + (1-\beta_2) dw^2$$
$$S_{db} = \beta_2 S_{db} + (1-\beta_2) db^2$$
$$V_{dw}^{current} = V_{dw} / (1-\beta_1^t), \quad V_{db}^{current} = V_{db} / (1-\beta_1^t)$$
$$S_{dw}^{current} = S_{dw} / (1-\beta_2^t), \quad S_{db}^{current} = S_{db} / (1-\beta_2^t)$$

$$W := W - \alpha \frac{V_{dW}^{current}}{\sqrt{S_{dW}^{current}} + \varepsilon} \qquad b := b - \alpha \frac{V_{db}^{current}}{\sqrt{S_{db}^{current}} + \varepsilon}$$

hyperparameters choice:

α: need to be tune

β_1: 0.9 → (dW)

β_2: 0.999 → (dW^2)

ε: 10^{-8}