Week 2 - Optimization algorithms

Friday, August 14, 2020 11:26 PM

① Mini-batch gradient descent

→ Gradient descent requires all training examples to be processed to start training

→ This slows down process when data is extremely large

→ Hence, mini-batches are used

eg: if $m = 5,000,000$ (5 million)

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

↓ mini batch

$Y = [\underbrace{y^{(1)}\ y^{(2)} \dots y^{(1000)}}_{Y^{\{1\}}\,(1,1000)} \mid \underbrace{y^{(1001)} \dots y^{(2000)}}_{Y^{\{2\}}\,(1,1000)} \mid \dots \mid \underbrace{\dots y^{(m)}}_{Y^{\{5,000\}}}]$

→ Mini-batch gradient descent

Mini-batch gradient 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 (1000 examples)

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

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

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

}

"1 epoch" — pass through training set.

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

X, Y

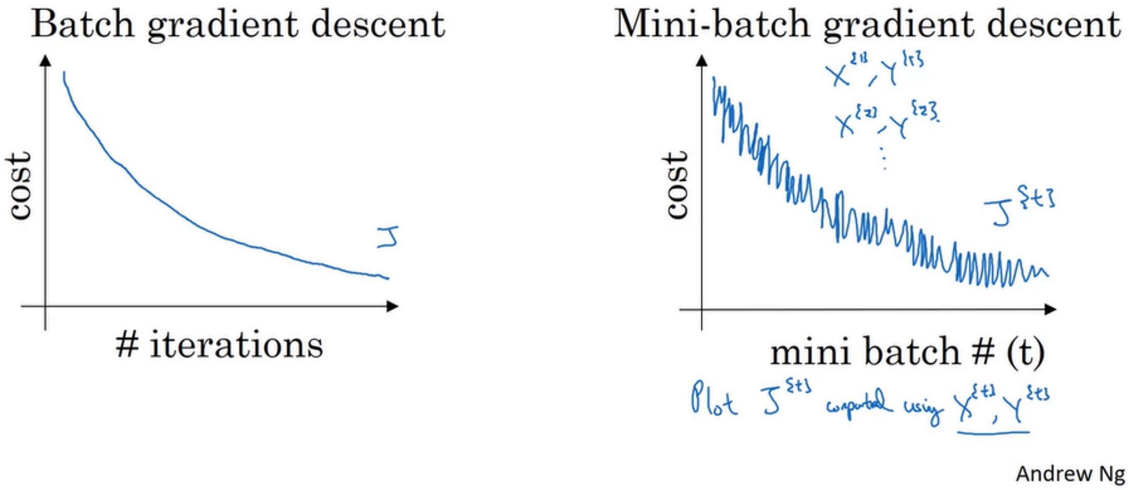
Andrew Ng

② Understanding mini-batch gradient descent

→ Training with mini batch gradient descent

Training with mini batch gradient descent

Training with mini batch gradient descent



Andrew Ng

→ Choosing mini-batch size

Too long per iteration

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

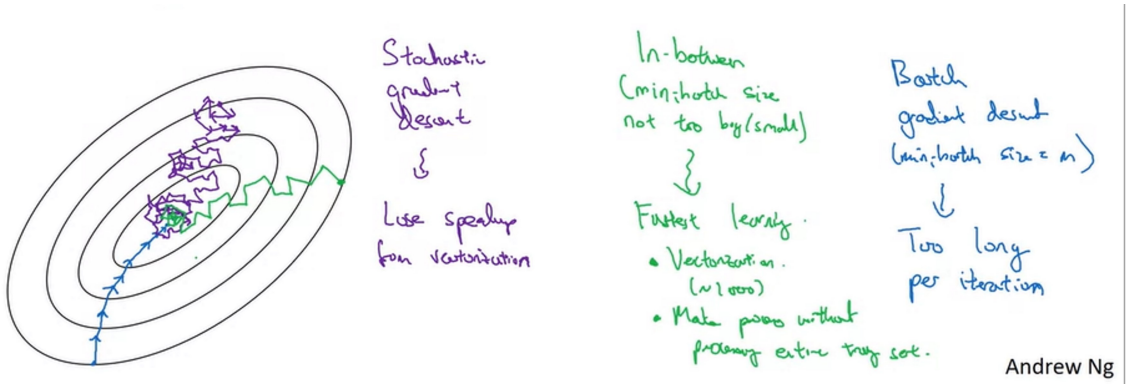
never reaches optimum but revolves around it

- If mini-batch size = 1: Stochastic gradient descent $(x^{\{t\}}, y^{\{t\}}) = (x^{[1]}, y^{[1]}) \dots (x^{[m]}, y^{[m]})$

↳ lose all speed benefits of vectorization

- In practice: between 1 & m

↳ Faster learning & vectorization



Andrew Ng

→ • If small training set: Use batch gradient descent ($m \leq 2000$)

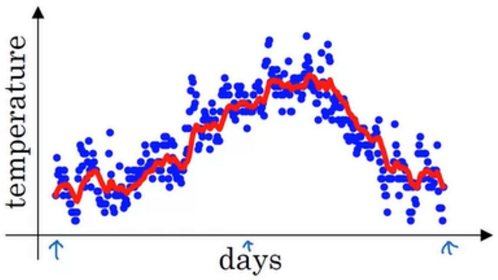
- If large training set:
  typical mini-batch size:
  64, 128, 256, 512 ...

make sure minibatch fits in CPU/GPU memory

(3) Exponentially weighted averages

Temperature in London

$\theta_1 = 40°F$ 4°C ←
$\theta_2 = 49°F$ 9°C
$\theta_3 = 45°F$ ⋮
⋮
$\theta_{180} = 60°F$ 15°C
$\theta_{181} = 56°F$ ⋮
⋮



$V_0 = 0$
$V_1 = 0.9\, V_0 + 0.1\, \theta_1$
$V_2 = 0.9\, V_1 + 0.1\, \theta_2$
$V_3 = 0.9\, V_2 + 0.1\, \theta_3$
⋮
$V_t = 0.9\, V_{t-1} + 0.1\, \theta_t$

Andrew Ng

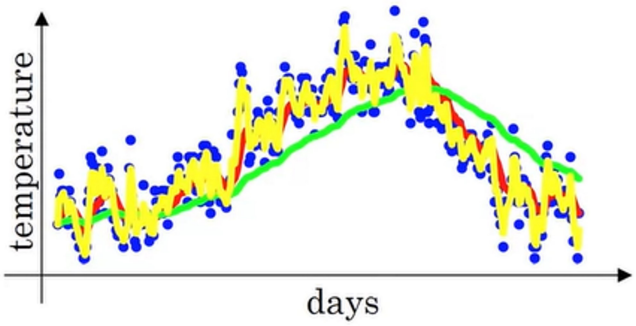
$$V_t = \beta V_{t-1} + (1-\beta)\theta_t$$ ($\beta = 0.9$ in above case)

V_t is approximately average over $\approx \frac{1}{1-\beta}$ days

$\therefore$ for $\beta = 0.9$, $\frac{1}{0.1} \Rightarrow 10$ days moving average

for $\beta = 0.98$, 50 days moving average

for $\beta = 0.5$, 2 day moving avg.



(4) Understanding exponentially weighted averages

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

Consider

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

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

$= 0.1\theta_{100} + 0.1(0.9)\theta_{99} + 0.1(0.9)^2\theta_{98} + 0.1(0.9)^3\theta_{97}$
$+ 0.1(0.9)^4\theta_{96} + \ldots\ldots$

↳ all coeff add upto ≈ 1

- when β^n reaches $\approx \frac{1}{e} \approx 0.36$ it is insignificant

$\therefore \beta = 0.9 \Rightarrow 0.9^{10} \approx 0.35$

- when $\beta^{..}$ reaches $\approx \frac{1}{e} \approx 0.36$ it is insignificant

$\therefore \beta = 0.9 \Rightarrow 0.9^{10} \approx 0.35$

$\therefore$ 10 days of values are significant

$\beta = 0.98 \Rightarrow 0.98^{50} \approx 0.36$

$\therefore$ 50 days of values are significant

This is how we get $\frac{1}{1-\beta}$ values formula

↳ general rule of thumb

- Implementation

$v_0 = 0$

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

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

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

...

→ one variable is enough and reused

$v_\theta := 0$

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

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

⋮

$\Rightarrow v_\theta = 0$

Repeat {

get next θ_t

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

}

(5) Bias correction in exponentially weighted average

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

$v_0 = 0$

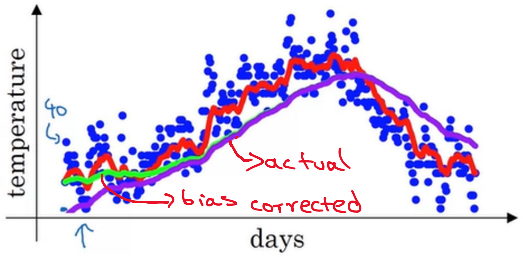
$v_1 = \cancel{0.98 v_0} \to 0 + 0.02\theta_1$

$\Rightarrow v_1 = 0.02\theta_1 \rightarrow$ very low

$v_2 = 0.98 v_1 + 0.02\theta_2$

$= 0.98 \times 0.02\theta_1 + 0.02\theta_2$

$= 0.0196\theta_1 + 0.02\theta_2 \rightarrow$ low



↑ compared to actual values θ_t

→ Bias correction:

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

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

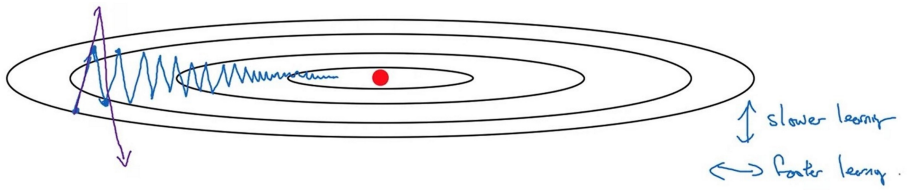
$\frac{v_2}{0.0396} = \frac{0.0196\theta_1 + 0.02\theta_2}{0.0396}$

When t gets larger, β^t gets smaller

↳ $1-\beta^t \approx 1$

$\therefore$ No effect on large values of t

⑥ Gradient descent with momentum

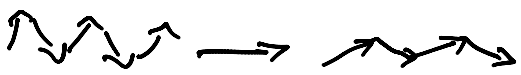


Momentum:
$v_{dw}=0,\ v_{db}=0$
On iteration t:

Compute dw, db on current mini-batch
$v_{dw} = \beta v_{dw} + (1-\beta)\, dw \quad (v_\theta = \beta v_\theta + (1-\beta)\theta_t)$
$v_{db} = \beta v_{db} + (1-\beta)\, db$
$w := w - \alpha v_{dw},\ b := b - \alpha v_{db}$

$v_{dw} = \beta v_{db} + dw$
↳ also used

↓
helps in dampening oscillations



On iteration t:

Compute dW, db on the current mini-batch

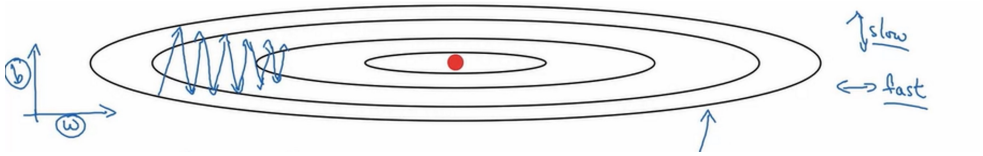
$v_{dW} = \beta v_{dW} + (1-\beta) dW$

$v_{db} = \beta v_{db} + (1-\beta) db$

$W = W - \alpha v_{dW},\ b = b - \alpha v_{db}$

- Hyperparameters: α, β
  α: l.r
  β: usually 0.9
- Bias correction is mostly not used since the bias affects only 10 initial iterations, insignificant on a huge set

⑦ RMSprop



on iteration t:

Comput dw, db on current mini-batch

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

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

$\epsilon = 10^{-8}$

$S_{dw} = \beta_2 S_{dw} + (1-\beta_2) dw$ ← small

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

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

$\varepsilon \to 10^{-8}$ → ensures not divided by zero

dividing by small no. ⇒ amplifies horizontal mvt.

dividing by large no. ⇒ dampens vertical mvt.

(8) 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

$v_{dw} = \beta_1 v_{dw} + (1-\beta_1) dw$, $v_{db} = \beta_1 v_{db} + (1-\beta_2) db$ → Momentum

$S_{dw} = \beta_2 S_{dw} + (1-\beta_2) dw$, $S_{db} = \beta_2 S_{db} + (1-\beta_2) db$ → RMSprop

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

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

$w := w - \alpha \frac{v_{dw}^{corrected}}{\sqrt{S_{dw}^{corrected}} + \varepsilon}$, $b := b - \alpha \frac{v_{db}^{corrected}}{\sqrt{S_{db}^{corrected}} + \varepsilon}$

→ Hyperparameters choice:

α : needs tuning

β_1 : 0.9 $\quad (dw)$

β_2 : 0.999 $\quad (dw^2)$

ε : 10^{-8}

} given in original paper published for ADAM (adaptive moment estimation)

(9) Learning rate decay

→ During initial steps large learning rate can be used

→ But as algorithm reaches convergence large α might cause noise or overshooting

→ Hence we implement decay of learning rate every epoch

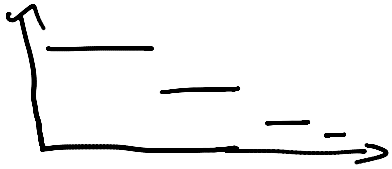
$\alpha = \frac{1}{1 + \text{decay rate} * \text{epoch-num}} \alpha_0$

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

manual decay for training over long time (days)

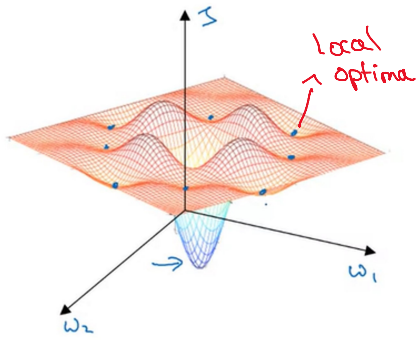
other formulas: $0.95^{\text{epoch-num}} \cdot \alpha_0$

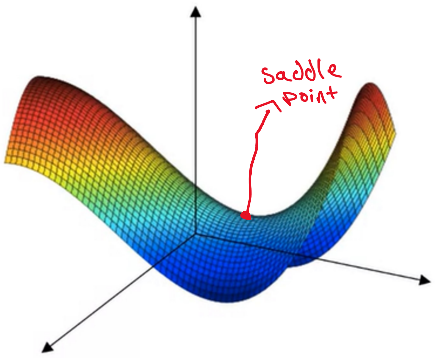
$\alpha = \frac{k}{\sqrt{\text{epoch-num}}} \cdot \alpha_0$ (or) $\frac{k}{\sqrt{t}} \alpha_0$



discrete staircase

(10) The problem of local Optima





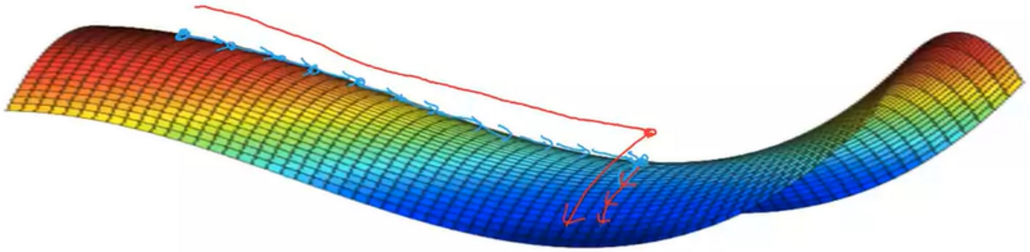
→ In high dimensional space local optimas are highly improbable

→ saddle points may occur

→ Problem of plateaus
- Gradient is zero for a long time

→ Intuitions that we have about space aren't applicable in the high dimensional space where we compute optimas for neural networks

Problem of plateaus



- Unlikely to get stuck in a bad local optima
- Plateaus can make learning slow